Capítulo 5

Evaluación mediante cargas de trabajo

El campo de la arquitectura de ordenadores es cada vez más cuantitativo. Las características de diseño se adoptan, únicamente, después de realizar evaluaciones detalladas de las ventajas y desventajas. Una vez que se han construido los sistemas, se evalúan y comparan tanto por arquitectos, para entender las ventajas y desventajas del diseño, como por usuarios, para tomar decisiones de compra. En diseños uniprocesador, una rica base de máquinas existentes y de aplicaciones ampliamente usadas soportan el proceso de identificación y evaluación de las decisiones de diseño. Los diseñadores aislan las características del rendimiento en las máquinas usando microbenchmarks — pequeños programas que fuerzan una característica determinada de la máquina. Los benchmarks más populares se codifican en un conjunto de benchmarks estándar, como el conjunto de benchmarks SPEC (Standard Performance Evaluation Corporation) para cargas de trabajo de ingeniería, y las medidas se realizan sobre el abanico de alternativas de diseño existentes. Basándose en estas medidas, en valoraciones sobre la tecnología emergente, y los cambios esperados en las necesidades de las aplicaciones, los diseñadores proponen nuevas alternativas. Aquellas que parecen prometedoras se evalúan, normalmente, mediante simulación. En primer lugar, se escribe un *simulador* — un programa que simula el diseño con y sin la característica de interés. A continuación se eligen un conjunto de cargas de trabajo de programas o de multiprogramación, ya sea de las *suites* estándares o de otras cargas de trabajo representativas de aquellas que se van a ejecutar sobre la máquina. Estas cargas de trabajo se ejecutan en el simulador y se determina el impacto en el rendimiento de la característica bajo estudio. Esto, junto con el coste estimado de la característica en tiempo de diseño y hardware, determina la inclusión de la misma. Los simuladores se escriben para que sean flexibles, de tal manera que pueda variarse los parámetros de la organización y del rendimiento para entender su impacto.

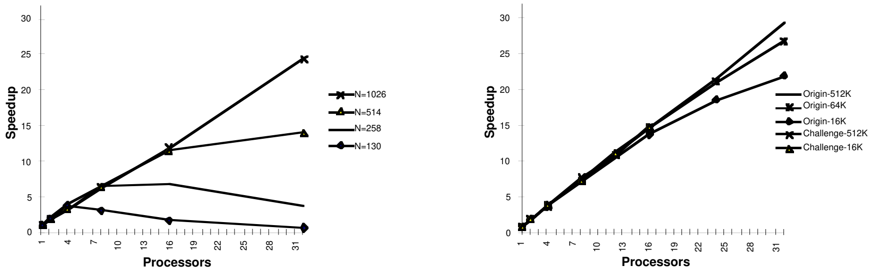
Una buena evaluación mediante cargas de trabajo es un proceso largo y difícil, incluso para sistemas uniprocesador. Las cargas de trabajo deben renovarse cuando cambian los patrones de uso y la tecnología. Las suites estándar de benchmarks se revisan cada pocos años. En particular, el conjunto de entradas usadas por los programas afecta a muchas de las interacciones claves en los sistemas y determina que se hagan visibles las características más importantes del sistema. Estas interacciones deben entenderse y reflejarse en el uso de las cargas de trabajo. Por ejemplo, para tener en cuenta el gran incremento en las velocidades de los procesadores y los cambios en los tamaños de las cachés, uno de los cambios más importantes de SPEC92 a SPEC95 fue usar un conjunto de datos de entrada mayor para acentuar el trabajo del sistema de memoria. Además, los simuladores precisos son costosos de desarrollar y verificar, y las simulaciones consumen una gran cantidad de tiempo de computación. Sin embargo, estos esfuerzos se ven bien recompensados ya que las buenas evaluaciones dan lugar a buenos diseños.

Con la maduración de las arquitecturas multiprocesador, se ha adoptado una aproximación cuantitativa similar. Mientras que las primeras máquinas paralelas eran, en muchos casos, audaces trabajos de artesanía, descansando básicamente en la intuición de los diseñadores, el diseño moderno incluye una evaluación considerable de las características de diseño propuestas. También aquí, las cargas de trabajo se usan para evaluar las máquinas reales, extrapolar los diseños propuestos y explorar su rendimiento a través de la simulación software. Para multiprocesadores, las cargas de trabajo de interés son o bien programas paralelos, o bien mezclas multiprogramadas de programas secuenciales y paralelos. La evaluación es una parte crítica de la nueva aproximación ingenieril a las arquitecturas multiprocesador; es muy importante entender los puntos claves de la evaluación antes de examinar el núcleo de las arquitecturas multiprocesador o la evaluación de los beneficios/costes que veremos en los temas siguientes.

Desafortunadamente, la evaluación basada en cargas de trabajo para arquitecturas multiprocesador es incluso más complicada que en el caso de los uniprocesadores. Esto se debe a varias razones:

- *Falta de maduración en las aplicaciones paralelas*. No es fácil de obtener unas cargas de trabajo "representativas" para los multiprocesadores, debido tanto a que su uso está relativamente inmaduro, como a que existen muchas nuevas características de comportamiento que deben considerarse.
- *Falta de maduración de los lenguajes de programación paralelos*. El modelo software para la programación paralela no se ha estabilizado, y los programas escritos tienen comportamiento muy diferentes dependiendo del modelo utilizado.
- *Sensibilidad en las diferencias de comportamiento*. Diferentes cargas, e incluso diferentes decisiones hechas en la paralelización de la misma carga secuencial, pueden presentar características de ejecución muy diferentes con respecto a una arquitectura.
- *Nuevos grados de libertad*. Aparecen varios nuevos grados de libertad en la arquitectura. El más obvio es el número de procesadores. Otros incluyen los parámetros de organización y rendimiento de la jerarquía de memoria extendida, particularmente la arquitectura de la comunicación. Junto con los grados de libertad de las cargas de trabajo (e.d., los parámetros de la aplicación) y del nodo uniprocesador subyacente, estos parámetros dan lugar a un espacio de diseño muy grande para la experimentación, particularmente cuando se evalúa una idea o un coste/beneficio en un contexto general, en lugar de evaluar una máquina fija. El alto coste de la comunicación hace que el rendimiento sea mucho más sensible a las interacciones entre todos estos grados de libertad que en el caso de los uniprocesadores, haciendo que sea más importante entender como conducir este gran espacio parametrizado.
- *Limitaciones en la simulación*. La simulación software de multiprocesadores en la evaluación de decisiones de diseño es más intensiva en recursos que la simulación de uniprocesadores. Las simulaciones de multiprocesadores consumen mucha memoria y tiempo. Por lo tanto, aunque el espacio de diseño que queremos explorar es mayor, el espacio que podemos explorar realmente es, a menudo, mucho más pequeño, y debemos de ser cuidadosos al decidir que partes del espacio vamos a simular.

En este capítulo aprenderemos que la evaluación efectiva necesita de la comprensión de propiedades importantes, tanto de las cargas de trabajo como de las arquitecturas, así como de la comprensión de



(a) Ocean sobre Origin2000

(b) Barnes-Hut sobre dos máquinas

Figura 5.1: Impacto de los parámetros de la aplicación en el rendimiento paralelo.

las interacciones existentes entre estas propiedades. En particular, las relaciones entre los parámetros de las aplicaciones y el número de procesadores determinan propiedades fundamentales del programa como el ratio comunicación/cálculo, distribución equilibrada de la carga, y localidad temporal y espacial. Estas propiedades interactúan con los parámetros de la jerarquía de memoria extendida para influir en el rendimiento en función de la aplicación y a menudo de manera drástica (ver figura 5.1). La elección de los valores de la carga de trabajo y de la máquina (o *tamaños*), así como la comprensión de las relaciones cuando varía dicho tamaño es un aspecto crucial de la evaluación basada en cargas de trabajo.

5.1 Escalamiento de las cargas de trabajo y de las máquinas

Comencemos por discutir algunas medidas básicas de rendimiento sobre un multiprocesador para a partir de este punto mostrar la importancia de un escalamiento apropiado, antes de ver las implicaciones de los distintos modelos de escalamiento vistos en la sección 1.5.

5.1.1 Medidas básicas del rendimiento de los multiprocesadores

Supongamos que hemos elegido un programa paralelo como carga de trabajo y queremos usarlo para evaluar una máquina. Para una máquina paralela, podemos medir dos características de rendimiento: el *rendimiento absoluto* y la *mejora en el rendimiento debido al paralelismo*. Este último se mide, normalmente, como el speedup, que se definió en la sección 1.4 como el rendimiento absoluto conseguido con p procesadores dividido por el que se consigue sobre un único procesador. El rendimiento absoluto (junto con el coste) es la medida más importante para el usuario final o para el comprador de una máquina. Sin embargo, por sí mismo, esta medida no nos dice gran cosa acerca de qué parte del rendimiento proviene del uso del paralelismo y de la efectividad de la arquitectura de las comunicaciones en comparación con el rendimiento del nodo uniprocesador que hay debajo. El Speedup nos dice que parte del rendimiento viene del uso del paralelismo; pero hay que tener en cuenta que es más fácil conseguir buenos speedup cuando los nodos individuales tienen un rendimiento menor ya que los costos de comunicación tienen un menor peso cuando el cálculo es más lento. Ambas métricas son importantes, y ambas deben medirse.

El rendimiento absoluto se mide mejor como trabajo realizado por unidad de tiempo. Dado un programa, la cantidad de trabajo a realizar se define normalmente mediante la configuración de entrada sobre la cual opera el programa, también denominada *tamaño del problema* (posteriormente definiremos el tamaño del problema de forma más precisa). Supongamos que la configuración de la entrada, y por tanto el trabajo, se mantiene fijo para un conjunto de experimentos. Entonces, podemos tratar el trabajo como un punto de referencia fijo, medir el tiempo de ejecución, y definir el rendimiento como el recíproco del tiempo de ejecución.

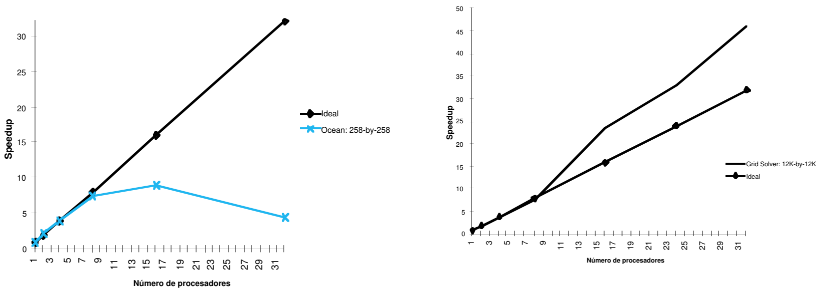
En algunos dominios de aplicación, los usuarios encuentran más conveniente tener una representación explícita, y dependiente del dominio, del trabajo y usar una métrica del rendimiento de trabajo por unidad de tiempo incluso cuando la configuración de entrada está fija. Por ejemplo, en un sistema de procesamiento por transacciones, la métrica puede ser el número de transacciones servidas por minuto; en una aplicación de ordenación, el número de claves ordenadas por segundo; en una aplicación de química, el número de enlaces calculados por segundo. Sin embargo, incluso aunque el trabajo se representa explícitamente, el rendimiento se mide con referencia a una configuración de entrada o cantidad de trabajo y, no obstante, estas métricas de rendimiento derivan de medidas del tiempo de ejecución (junto con el número de eventos de interés de la aplicación). Dada una configuración del problema fija y conocida, estas métricas específicas del dominio no presentan una ventaja fundamental sobre el tiempo de ejecución o su recíproco. De hecho, debemos ser cautelosos para asegurarnos que la medida explícita de trabajo utilizada es además una medida con significado desde la perspectiva de la aplicación, no algo con lo que podamos hacer trampas. Posteriormente discutiremos las propiedades deseables de las métricas de trabajo cuando consideremos en mayor detalle dichas métricas en la sección 5.2.5. Por ahora, nos centraremos en la evaluación de la mejora en el rendimiento absoluto debido al paralelismo, es decir, el speedup debido al uso de p procesadores en lugar de uno.

5.1.2 ¿Por qué preocuparse por el escalamiento?

Desafortunadamente, existen varias razones por las cuales la medición del speedup (o aceleración) para un tamaño de problema fijo es insuficiente como la *única* manera de evaluar la mejora en las prestaciones debido al paralelismo para un conjunto máquinas de diferente tamaño.

Supongamos que el problema de tamaño fijo que hemos elegido es relativamente pequeño y apropiado para una máquina con pocos procesadores. Al incrementar el número de procesadores para el mismo tamaño de problema, las sobrecargas debidas al paralelismo (comunicación, distribución desequilibrada de la carga) incrementan su peso relativo con respecto al cálculo útil. Llegará el momento en que el tamaño del problema sea demasiado pequeño para que la evaluación de la máquina con el mismo sea realista. Las altas sobrecargas dan lugar a pequeñas aceleraciones que no reflejan tanto las capacidades de la máquina como el hecho de que estemos usando un tamaño de problema inapropiado (es decir, uno que no tiene suficiente concurrencia para la máquina grande). De hecho, en determinado momento añadir más procesadores puede perjudicar el rendimiento al dominar la sobrecarga al trabajo útil (ver figura 5.2a). Un usuario no ejecutaría este problema en una máquina tan grande, por lo que es inapropiado para evaluar dicha máquina.

Por otra parte, si elegimos un problema que es realista para una máquina con muchos procesadores, podemos tener el problema opuesto al evaluar la mejora de rendimiento debida al paralelismo. Este problema puede ser demasiado grande para un único procesador debido a que los datos sean tan



(a) Problema Ocean pequeño

(b) Speedup superlineal con un problema de resolución de ecuaciones grande

Figura 5.2: Speedups en el SGI Origin 2000 cuando se incrementa el número de procesadores.

grandes que desborden la memoria de un único nodo. En algunas máquinas, el problema no se podrá si quiera ejecutar; en otras, aparecerán problemas de trasiego de páginas al disco duro. Cuando se utilicen suficientes procesadores, los datos cabrán en la memoria de los mismos, eliminando esta comunicación artificial. Los cálculos dentro de cada nodo será más eficientes y como resultados obtendremos un speedup (o aceleración) muy por encima del número de procesadores usados. Una vez que ha ocurrido este hecho, las futuras mejoras en el speedup se comportarán de una manera más usual conforme se incremente el número de procesadores, pero la aceleración con respecto a un único procesador será todavía *superlineal*.

Una última limitación de mantener el mismo tamaño de problema cuando se incrementa el número de procesadores es que este hecho no refleja el uso real de la máquina tal como vimos en la sección 1.5.

Escalar una máquina significa hacerla más (o menos) potente. Esto se puede conseguir haciendo que cualquier componente de la máquina sea mayor, más sofisticado, o más rápido — los procesadores individuales, las cachés, la memoria, la arquitectura de comunicación o el sistema de E/S. En general, el *tamaño de la máquina* es un vector que caracteriza las capacidades de cada nodo de procesamiento, de la jerarquía de memoria, las comunicaciones y la E/S. Escalar una máquina significa cambiar una o más entradas en este vector. Dado que nuestro interés es el paralelismo, definimos el tamaño de una máquina como el número de procesadores y supondremos que cada nodo individual permanece inalterable cuando se escala la máquina. Por ejemplo, escalar una máquina con p procesadores y $p \times m$ megabytes de memoria total por un factor de k, da como resultado una máquina con $k \times p$ procesadores y $k \times p \times m$ megabytes de memoria total.

El *tamaño del problema* se refiere a una instancia específica o configuración de la entrada. Dicha configuración viene dada, normalmente, mediante un vector de parámetros de entrada. Por ejemplo, el tamaño del problema en la aplicación Ocean viene dado por un vector $V = (n, \varepsilon, \Delta t, T)$, donde n es el tamaño de la malla en cada dimensión (que especifica la resolución espacial de nuestra representación del problema), ε es el error que toleramos para determinar la convergencia de la resolución de ecuaciones en una multimalla, Δt es la resolución temporal (e.d., el tiempo físico entre pasos de tiempo), y T es el número de pasos de tiempo realizados.

El *tamaño del problema* debe distinguirse del tamaño del conjunto de datos. El *tamaño del conjunto de datos* es la cantidad de almacenamiento necesaria para ejecutar el programa sobre un procesador. Este, a su vez, es distinto del *uso de memoria* del programa, que es la cantidad de memoria usada por el programa paralelo, incluyendo la replicación de datos. El tamaño del conjunto de datos depende, normalmente, de un pequeño número de parámetros del programa. En Ocean, por ejemplo, viene dado únicamente por el tamaño de la malla, n. Sin embargo, el número de instrucciones y el tiempo de ejecución depende del resto de parámetros del problema. Así, mientras que el vector del tamaño del problema, V, determina muchas propiedades importantes del programa —como el tamaño del conjunto de datos, el número de instrucciones que ejecuta y el tiempo de ejecución— no es idéntico a ninguna de estas propiedades.

5.1.3 Cuestiones claves en el escalamiento

Dadas estas definiciones, existen dos cuestiones principales a tener en cuenta cuando se escala un problema para su ejecución en una máquina más grande:

1. ¿Bajo qué limitaciones debe escalarse el problema?. Para definir un modelo de escalamiento, algunas propiedades deben mantenerse fijas cuando se escala la máquina. Estas propiedades, como vimos en la sección 1.5, pueden incluir el tamaño del conjunto de datos, la memoria usada por el procesador, el tiempo de ejecución, el número de transacciones ejecutadas por segundo, el número de partículas o filas de una matriz asignadas a cada procesador, etc.

2. ¿Cómo debemos escalar el problema?. Es decir, ¿cómo deben cambiar los parámetros en el vector del tamaño del problema para conseguir las restricciones elegidas?.

5.1.4 Impacto de los modelos de escalamiento en el núcleo de resolución de ecuaciones

Usemos un ejemplo simple, el núcleo de resolución de ecuaciones, para ver su interacción con los diferentes modelos de escalamiento y como éstos afectan a las características de comportamiento relevantes desde el punto de vista arquitectónico. Veamos en que consiste este núcleo.

El núcleo de resolución de ecuaciones resuelve una ecuación diferencial parcial simple sobre una malla, usando el método de diferencias finitas. Opera sobre una malla bidimensional regular de $(n + 2) \times (n + 2)$ elementos. Las filas y columnas del borde contienen valores frontera que no cambian, mientras que los $n \times n$ puntos del interior se actualizan por el programa, empezando por sus valores iniciales. El cálculo se realiza durante un número de ciclos. En cada ciclo, se opera sobre todos los puntos interiores de la malla. Para cada punto, se reemplaza su valor con la media ponderada de él mismo y de sus cuatro puntos vecinos más cercanos —arriba, abajo, izquierda y derecha (ver figura 5.3). Las actualizaciones se realizan sobre la malla, de tal manera que el cálculo de actualización para un punto ve los nuevos valores de sus puntos de arriba y de la izquierda del mismo, y los valores antiguos de su punto de abajo y de la derecha. Esta forma de actualización se denomina el método de Gauss-Seidel. Durante cada iteración, el núcleo también calcula la diferencia media entre el elemento actualizado y su valor anterior. Si la diferencia media en todos los elementos es menor que un parámetro de tolerancia predefinido, se dice que la solución ha convergido y el procedimiento termina. En caso contrario, se realiza otra iteración y se comprueba de nuevo la convergencia.

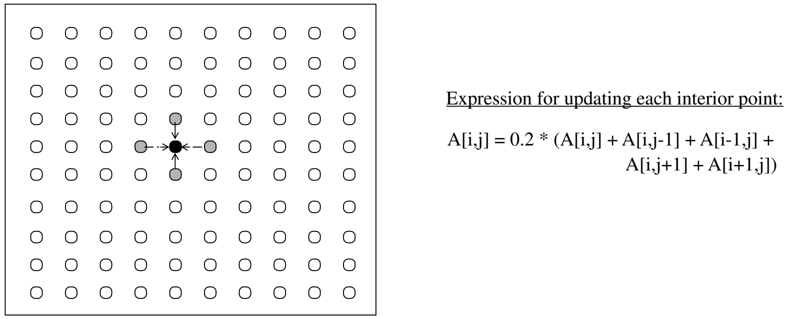


Figura 5.3: Actualización del vecino más cercano en un punto de la malla de la resolución de ecuaciones simple.

Para una malla $n \times n$, los requisitos de memoria para el núcleo de resolución de ecuaciones simple es $O(n^2)$. La complejidad computacional es $O(n^2)$ veces el número de iteraciones que tarda en converger, que podemos suponer de forma conservadora que es $O(n)$ (el número de iteraciones necesarias para que los valores fluyan de un lado de la malla al otro). Esto da lugar a un complejidad computacional en la implementación secuencial de $O(n^3)$.

Consideremos el tiempo de ejecución y las necesidades de memoria bajo los tres modelos de escalamiento, suponiendo aceleraciones debidas al paralelismo iguales al número de procesadores en todos los casos. Con el escalamiento con carga de trabajo fija, conforme se divide la misma malla de $n \times n$ entre más procesadores p, las necesidades de memoria por procesador decrece linealmente con p, al igual que el tiempo de ejecución. Bajo el modelo de limitación del tiempo, el tiempo de ejecución permanece igual por definición. Suponiendo un speedup lineal, esto significa que si el tamaño de la malla escalada es $k \times k$, entonces $\frac{k^3}{p} = n^3$, con lo que $k = n \times p^{\frac{1}{3}}$. La cantidad de memoria necesaria por procesador es, por lo tanto

$$\frac{k^2}{p} = \frac{n^2}{p^{\frac{1}{3}}}$$

que disminuye con la raíz cúbica del número de procesadores. Usando el escalamiento de restricción de la memoria, por definición, las necesidades de memoria por procesador permanecen igual a $O(n^2)$ donde la malla base para la ejecución de un único procesador es $n \times n$. Esto significa que el tamaño total de la malla se incrementa por un factor de p, por lo que el tamaño de la malla es ahora de $n\sqrt{p} \times n\sqrt{p}$ en lugar de $n \times n$. Dado que ahora tarda $n\sqrt{p}$ iteraciones en converger, la complejidad del tiempo secuencial es $O((n\sqrt{p})^3)$. Esto significa que incluso suponiendo un speedup perfecto debido al paralelismo, el tiempo de ejecución para el problema escalado sobre p procesadores es

$$O\left(\frac{(n\sqrt{p})^3}{p}\right)$$

o $n^3\sqrt{p}$. Por lo tanto, el tiempo de ejecución paralelo es mayor que el tiempo de ejecución secuencial del problema base por un factor de $\sqrt{p}$. Incluso bajo la suposición de un speedup lineal, un problema

que tarda una hora en un procesador necesitará 32 horas en un máquina con 1024 procesadores bajo el escalamiento restringido por la memoria. Para esta caso, el tiempo de ejecución se incrementa rápidamente bajo el escalamiento de restricción de la memoria, y las necesidades de memoria por procesador decrecen bajo el escalamiento restringido por el tiempo.

Consideremos el efecto de los diferentes modelos de escalamiento en la concurrencia, ratio comunicación/cálculo, frecuencia de sincronización y E/S, localidad temporal y espacial, y tamaño de los mensajes (en el modelo de paso de mensajes).

La concurrencia de este núcleo es proporcional al número de puntos en la malla. Este valor permanece fijo bajo el escalamiento con tamaño de problema fijo, crece proporcionalmente a p bajo el escalamiento de memoria fija, y crece proporcionalmente a $p^{0.67}$ bajo el escalamiento con tiempo fijo.

El ratio comunicación/cálculo es el ratio perímetro/área de la partición de la malla asignada a cada procesador; es decir, es inversamente proporcional a la raíz cuadrada del número de puntos por procesador $\left(\frac{n^2}{p}\right)$. Bajo el escalamiento de tamaño fijo, el ratio crece proporcionalmente a $\sqrt{p}$. Bajo el escalamiento de memoria fija, el tamaño de una partición no cambia, por lo que tampoco lo hace el ratio comunicación/cálculo. Finalmente, bajo el escalamiento con tiempo fijo, dado que el tamaño de la partición asignada a un procesador disminuye con la raíz cúbica del número de procesadores, el ratio se incrementa proporcionalmente a la raíz sexta de p.

El núcleo de resolución de ecuaciones se sincroniza al final de cada fase para determinar la convergencia. Supongamos que la E/S también se realiza en ese punto, por ejemplo, mostrando el máximo error al final de cada iteración. Bajo el escalamiento de tamaño fijo, el trabajo realizado por cada procesador en una iteración dada decrece linealmente cuando se incrementa el número de procesadores. Así, suponiendo un speedup lineal, la frecuencia de sincronización y E/S crece linealmente con p. Bajo el escalamiento de memoria fija la frecuencia permanece fija, y bajo el escalamiento de tiempo fijo se incrementa con la raíz cúbica de p.

El tamaño de los conjuntos de trabajo importantes, que indica su localidad temporal, en este caso es exactamente el tamaño de la partición de la malla para un procesador. Por lo tanto, el tamaño de la partición y las necesidades de las cachés disminuyen linealmente con respecto a p bajo el escalamiento de tamaño fijo, permanece constante bajo el escalamiento de memoria fija y disminuye proporcionalmente a la raíz cúbica de p bajo el escalamiento de tiempo fijo. Así, aunque el tamaño del problema total crece bajo el escalamiento de tiempo fijo, el tamaño del conjunto de trabajo de cada procesador disminuye.

La localidad espacial del núcleo de resolución de ecuaciones es mejor dentro de la partición del procesador y en las fronteras orientadas por filas y peor en las fronteras orientadas por columnas. Así, decrece cuando la partición del procesador es más pequeña y las fronteras orientadas por columnas se hacen más grandes en relación con el área de la partición. Por lo tanto, permanece constante bajo el escalamiento de memoria fija, decreciendo rápidamente bajo el escalamiento de tamaño fijo y decrece menos rápidamente bajo el escalamiento de tiempo fijo.

Finalmente, un mensaje individual en un modelo de paso de mensajes es probable que sea una fila o columna exterior de la partición del procesador, que es la raíz cuadrada del tamaño de la partición. Por lo tanto, el tamaño del mensaje aumenta de forma similar al ratio comunicación/cálculo. Sin embargo, el número de mensajes que envía un proceso depende únicamente del número de procesadores vecinos y es independiente de n, p o modelo de escalamiento.

De esta discusión parece claro que siempre que los efectos de memoria o de capacidad en la caché no sean dominantes, podemos esperar la menor sobrecarga de paralelismo y el mayor speedup bajo el escalamiento de memoria fija y a continuación bajo el escalamiento de tiempo fijo. Debemos esperar una degradación rápida de los speedups bajo el escalamiento de tamaño del problema fijo, al menos una vez que las sobrecargas se hagan significativas en relación con el trabajo útil. También está claro que la elección de los parámetros de la aplicación y el modelo de escalamiento afecta de forma importante tanto a las características del programa fundamentales como a las interacciones arquitecturales con la jerarquía de memoria extendida, como la localidad espacial y temporal. Al menos que se sepa que un modelo de escalamiento particular es el correcto para una aplicación, o es particularmente inapropiado, es útil evaluar una máquina bajo los tres modelos de escalamiento. Examinaremos las interacciones con los parámetros arquitecturales y su importancia para la evaluación más detalladamente cuando discutamos evaluaciones reales (secciones 5.2 y 5.3).

5.1.5 Escalamiento de los parámetros de la carga

Al discutir las restricciones sobre las que deben escalarse el tamaño del problema, no se examinó cómo los diferentes parámetros de la aplicación que constituyen el vector del tamaño del problema deben escalarse de forma relativa entre ellos para conseguir la restricción elegida. Por ejemplo, la aplicación Ocean tiene un vector de cuatro parámetros: n, ε, Δt y T. Cómo deben escalarse los parámetros de la carga entre sí no es un problema bajo el modelo de escalamiento de tamaño de problema fijo, pero sí lo es en los otros dos casos. Los diferentes parámetros están a menudo relacionados entre sí, y puede no tener sentido escalar uno de ellos sin escalar los otros, o escalarlos de forma independiente. Por ejemplo, en un uso realista de la aplicación Barnes-Hut, el parámetro θ (la exactitud en el cálculo de fuerzas) y Δt (el intervalo físico entre pasos de tiempo) debe escalarse cuando cambia n (el número de cuerpos). Todos estos parámetros pueden contribuir a una característica de ejecución dada; por ejemplo, el tiempo de ejecución de Barnes-Hut crece proporcionalmente a

$$\frac{1}{\theta^2 \Delta t} n \log n$$

Como resultado de este hecho, el incremento en el número de cuerpos, n, bajo el escalamiento de tiempo fijo no es tan grande como el que se deduciría del escalamiento únicamente de n.

Incluso el núcleo de resolución de ecuaciones tiene otro parámetro, ε, que representa la tolerancia usada para determinar la convergencia del programa. Hacer esta tolerancia más pequeña — tal como debe hacerse cuando se escala n en una aplicación real— incrementa el número de iteraciones necesarias para que converga y, por tanto, incrementa el tiempo de ejecución, pero no afecta a las necesidades de la memoria. Comparado con el escalamiento únicamente de n, el escalamiento de ε y n hace que el tamaño de la malla por procesador, las necesidades de memoria y el tamaño del conjunto de trabajo decrezca mucho más rápidamente bajo el escalamiento de tiempo fijo, que el ratio comunicación/cálculo se incremente más rápidamente bajo el escalamiento de tiempo fijo pero que se mantenga fijo bajo el modelo de escalamiento de memoria fija, y el tiempo de ejecución se incrementa incluso más rápidamente bajo el escalamiento de memoria fija. Como arquitectos que usan cargas de trabajo, es muy importante que entendamos las relaciones entre los parámetros desde un punto de vista del usuario de la aplicación y que escalemos los parámetros en nuestras evaluaciones conforme a este entendimiento. En caso contrario, seremos responsables de llegar a conclusiones arquitecturales incorrectas.

Las relaciones reales entre los parámetros y las reglas para su escalamiento dependen del dominio y de la aplicación. No existen reglas universales. Por ejemplo, en aplicaciones como Barnes-Hut y Ocean que modelan fenómenos físicos a través de la discretización, los diferentes parámetros de la aplicación gobiernan diferentes fuentes de error en la exactitud con la que se modela el fenómeno (como la evolución de las galaxias); las reglas adecuadas para escalar estos parámetros conjuntamente vienen dadas por guías acerca de como escalan los diferentes tipos de error. Idealmente, las suites de benchmarks describirán las reglas de escalamiento de tal manera que el arquitecto no deba de preocuparse de aprender estas relaciones.

5.2 Evaluación de una máquina real

Ahora que entendemos la importancia de un escalamiento adecuado y los efectos que el tamaño del problema y de la máquina tienen sobre el comportamiento de las características fundamentales e interacciones con la arquitectura, estamos listos para desarrollar guías específicas para dos tipos de evaluación mediante cargas de trabajo: la evaluación de una máquina real y la evaluación de una idea relacionada con la arquitectura o *trade-off* en un contexto general. La evaluación de una máquina real es más simple en varios aspectos: la organización, granularidad y parámetros de rendimiento de la máquina están fijos, y sólo tenemos que preocuparnos de elegir adecuadamente las cargas de prueba y los parámetros de las mismas; además, no estamos restringidos por las limitaciones de la simulación software. En esta sección proporcionaremos una plantilla prescriptiva para la evaluación de una máquina real. Empezaremos con el uso de *microbenchmarks* para aislar las características relacionadas con el rendimiento. A continuación, mostraremos las principales puntos en la elección de una carga de trabajo — en primer lugar cuando el número de procesadores está fijo y a continuación cuando se permite que varíe. La sección finaliza con una discusión sobre las métricas más populares en la medición del rendimiento de una máquina y la presentación de los resultados de una evaluación.

5.2.1 Aislamiento del rendimiento utilizando Microbenchmarks

Un primer paso para evaluar una máquina real es el entendimiento de sus capacidades básicas de rendimiento — es decir, las características de rendimiento de las operaciones primitivas proporcionadas por el modelo de programación, la abstracción de las comunicaciones (interface usuario/sistema), o el interfaz hardware/software. Esto se consigue normalmente con pequeños programas especialmente escritos denominados *microbenchmarks* que se diseñan para aislar estas características de rendimiento (por ejemplo: latencia, anchos de banda, sobrecargas, etc.).

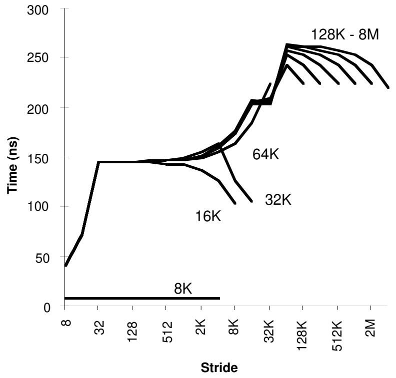
En los sistemas paralelos se usan cinco tipo de microbenchmarks; los tres primeros también se usan para la evaluación de uniprocesadores:

1. Los *microbenchmarks de procesamiento* miden el rendimiento del procesador sobre operaciones que no acceden a memoria, como las operaciones aritméticas, operaciones lógicas y saltos.
2. Los *microbenchmarks de memoria local* determinan la organización, latencia y ancho de banda de los niveles de la jerarquía de memoria dentro del nodo local y miden el rendimiento de las operaciones de lectura y escritura satisfecha a diferentes niveles, incluyendo aquellas que causan fallos en la TLB y fallos de página.
3. Los *microbenchmarks de entrada-salida* miden las características de las operaciones de E/S, tales como lecturas y escrituras a disco de varias longitudes y valores de salto (*stride*).

4. Los *microbenchmarks de comunicaciones* miden las operaciones de comunicación de datos, tales como envío y recepción de mensajes o lecturas y escrituras remotas de diferentes tipos.
5. Los *microbenchmarks de sincronización* miden el rendimiento de las diferentes operaciones de sincronización, tales como cerrojos y barreras.

Los microbenchmarks de comunicación y sincronización dependen de la abstracción de las comunicaciones o modelo de programación usado. Pueden incluir uno o un par de procesadores — por ejemplo, un fallo de lectura remoto, un par envío/recepción, o la adquisición de un cerrojo libre— o pueden ser colectivas, como el broadcast, reducción, comunicación todos con todos, patrones de comunicación probabilísticos, varios procesadores intentando adquirir un cerrojo, o barreras. Se pueden diseñar diferentes microbenchmarks para acentuar la latencia en ausencia de contención, el ancho de banda, la sobrecarga y la contención.

Con vistas a la medición, los microbenchmarks se suelen implementar como un conjunto de repeticiones de operaciones primitivas (p.e., 10000 lecturas remotas sobre una fila). Normalmente tienen parámetros simples que pueden variarse para obtener una caracterización más completa, por ejemplo, el número de procesadores que participan en un microbenchmark de comunicación colectiva o el salto entre dos lecturas consecutivas en un microbenchmark de memoria local. La figura 5.4 muestra un perfil típico de una máquina obtenido usando un microbenchmark de memoria local. La principal tarea de los microbenchmarks es aislar y entender el rendimiento de las capacidades básicas del sistema.



The Microbenchmark

```
for times ← 0 to10,000 do
    for i ← 0 to ArraySize-1 by stride do
        load A[i];
```

Figura 5.4: Resultado de un microbenchmark sobre un único nodo de procesamiento en el multiprocesador CRAY T3D.

Habiendo aislado las características del rendimiento, el siguiente paso es evaluar la máquina sobre cargas de trabajo más realistas. Debemos explorar tres ejes principales: las cargas de trabajo, el tamaño de sus problemas, y el número de procesadores (o el tamaño de la máquina). Los parámetros de bajo nivel de la máquina están fijos. Empecemos por la elección de las cargas de trabajo.

5.2.2 Elección de las cargas de trabajo

Más allá de los microbenchmarks, las cargas de trabajo se pueden dividir en tres clases en orden creciente del realismo y complejidad: núcleos, aplicaciones completas, y cargas de trabajo de multiprogramación. Cada una tiene su propio papel, ventajas y desventajas.

Los *núcleos* son partes bien definidas de aplicaciones reales pero no son aplicaciones completas por sí mismas. Pueden varias de simples núcleos, como una transposición de matrices , hasta núcleos más complejos y sustanciales que dominan el tiempo de ejecución de sus aplicaciones, como la factorización de matrices y los métodos iterativos que resuelven ecuaciones diferenciales parciales. Ejemplos de núcleos para el procesamiento de información incluyen complejas peticiones a bases de datos usadas en aplicaciones para soporte a la decisión o la ordenación de un conjunto de números. Los núcleos muestran interacciones de alto nivel que no están presentes en los microbenchmarks y, como resultado, se pierde en el grado de aislamiento del rendimiento. La propiedad clave es que sus características de rendimiento relevantes — ratio comunicación/cálculo, concurrencia y conjunto de trabajo, por ejemplo— pueden entenderse fácilmente y determinarse de forma analítica, de tal manera que el rendimiento observado como resultado de las interacciones puede explicarse a la luz de estas características.

Las *aplicaciones completas* consisten en múltiples núcleos y exhiben interacciones de mayor nivel entre los núcleos que un núcleo individual no puede revelar. Al contrario que los núcleos, las aplicaciones completas se ejecutan por usuarios para obtener una respuesta a lo que están buscando. Las mismas grandes estructuras de datos pueden accederse de diferentes formas por múltiples núcleos en una aplicación, y accesos a diferentes estructuras de datos por diferentes núcleos pueden interferirse mutuamente en la jerarquía de la memoria. Además, las estructuras de datos que son óptimas para un núcleo aislado pueden no ser la mejor para la aplicación completa. Lo mismo se cumple para las técnicas de particionamiento. Por ejemplo, si existen dos núcleos independientes en una aplicación, entonces podemos decidir no particionar cada uno entre todos los procesadores sino repartir los procesadores entre ellos. Diferentes núcleos que comparten una estructura de datos pueden dividirse de forma que se consiga un equilibrio entre sus diferentes accesos y patrones de comunicación, dando lugar a la máxima localidad total. La presencia de múltiples núcleos en una aplicación introduce muchas interacciones sutiles, y las características relacionadas con el rendimiento de las aplicaciones completas no se pueden determinar normalmente de forma analítica.

Las *cargas de trabajo multiprogramado* consisten en múltiples programas secuenciales y aplicaciones paralelas que se ejecutan conjuntamente en una máquina. Las diferentes aplicaciones pueden ser de tiempo compartido, de espacio compartido (e.d., diferentes aplicaciones ejecutándose sobre subconjuntos disjuntos de procesadores de la máquina), o de ambos a la vez, dependiendo de las políticas de multiprogramación del sistema operativo. Al igual que la evaluación de las aplicaciones completas son complicadas debido a interacciones de más alto nivel entre los núcleos que las componen, las cargas de trabajo multiprogramado incluyen complejas interacciones entre todas las aplicaciones.

Cuando nos movemos de los núcleos a aplicaciones completas y cargas de trabajo multiprogramado ganamos en realismo, lo que es muy importante. Muchos de los fallos críticos y problemas de rendimiento no se revelan en los microbenchmarks e incluso en los núcleos, pero se descubren con estas cargas de trabajo. Sin embargo, perdemos en la habilidad de describir de forma concisa la carga de trabajo, para explicar e interpretar los resultados de forma no ambigua, y aislar los factores del rendimiento. En el extremo, las cargas de trabajo multiprogramado son difíciles no solamente de interpretar sino también de diseñar: ¿qué aplicaciones deben de incluirse en esa carga de trabajo y en que proporción?. También es difícil obtener resultados repetibles de las cargas de trabajo multiprogramado debido a las interacciones con el sistema operativo que dependen sutilmente del tiempo. Cada tipo de carga tiene su lugar. Sin embargo, las interacciones de más alto nivel que aparecen únicamente con las aplicaciones completas y las cargas de trabajo multiprogramado (y el hecho de que son cargas que serán ejecutadas realmente sobre la máquina por parte de los usuarios) hace importante su uso para determinar finalmente el rendimiento total de una máquina.

5.2.3 Evaluación de una máquina de tamaño fijo

Habiendo fijado la carga de trabajo y el tamaño de la máquina, únicamente nos queda por elegir los parámetros de la carga de trabajo. Ya hemos visto que, para un número fijo de procesadores, cambiar el tamaño del problema puede afectar drásticamente a todas las características importantes de la ejecución y, por tanto, a los resultados de la evaluación. De hecho, incluso puede cambiar la naturaleza del cuello de botella dominante — es decir, si es la comunicación, la falta de una distribución equilibrada de la carga, o el acceso a los datos locales. Esto ya nos indica un punto muy significativo pero a menudo ignorado: normalmente no es suficiente con usar un único tamaño de problema al realizar una evaluación, incluso cuando el número de procesadores está fijo.

Podemos usar nuestro conocimiento acerca de las interacciones de la aplicación y la arquitectura para elegir tamaños de problema a estudiar. Nuestro objetivo es obtener una cobertura adecuada de los comportamientos inherentes y las interacciones arquitecturales al mismo tiempo que restringimos el número de tamaños de problema que necesitamos. Este objetivo se consigue mediante un conjunto de pasos estructurados, demostrando los peligros de elegir un único tamaño de problema en el proceso. En cada paso, se usará el núcleo de resolución de ecuaciones para ilustrar cuantitativamente los puntos. Para la aproximación cuantitativa supondremos que estamos evaluando una máquina de espacio de direcciones compartido con coherencia de caché y con 64 nodos de proceso, cada uno con 1 MB de caché y 64 MB de memoria principal. Los pasos son los siguientes.

Paso 1: Determinar el rango de los tamaños del problema

Una manera de elegir el tamaño de los problemas, aplicable en algunos casos afortunados, es el deseo de mayor potencia. Las metas de mayor nivel del estudio pueden elegir el tamaño de los problemas por nosotros. Por ejemplo, podemos saber que los usuarios de la máquina están interesados únicamente en unos pocos tamaños de problema específicos. Esto simplifica nuestro trabajo pero es poco común y no es una metodología de propósito general. Este caso no se aplica al núcleo de resolución de ecuaciones.

El conocimiento del uso real puede identificar un rango debajo del cual los problemas son pocos realistas en cuanto a su tamaño para la máquina que se evalúa y por encima del cual el tiempo de ejecución es demasiado grande o los usuarios no estarían interesados. Este caso tampoco es particularmente útil para el núcleo de resolución de ecuaciones. Una vez que hemos identificado un rango, podemos pasar al siguiente caso.

Paso 2: Usar las características de comportamiento inherentes

Las características de los comportamientos inherentes (como el ratio comunicación/cálculo y distribución de la carga) nos ayudan a restringir más aún el rango y a elegir tamaños de problema dentro del rango seleccionado. Dado que el ratio comunicación/cálculo inherente decrece normalmente con el incremento del tamaño del conjunto de datos, los problemas grandes pueden que no fuercen lo suficiente la arquitectura de comunicaciones —al menos con la comunicación inherente— mientras que los problemas pequeños pueden forzarla de forma no representativa y, potencialmente, ocultar otros cuellos de botella. Dado que la concurrencia se incrementa normalmente con el tamaño del conjunto de datos, nos gustaría elegir, al menos, algunos tamaños de problema que sean lo suficientemente grandes para se puedan balancear pero no tan grandes que la comunicación inherente sea demasiado pequeña. El tamaño del problema puede también verse afectado por las fracciones del tiempo de ejecución que se utiliza en las diferentes fases de la aplicación, que pueden tener distribuciones de carga, sincronización y características de comunicaciones muy diferentes. Por ejemplo, en el estudio de la aplicación Barnes-Hut, los problemas más pequeños están más tiempo en la fase de creación del árbol, que no se paraleliza muy bien y que tiene propiedades menos deseables, que en la fase de cálculo de fuerzas que domina usualmente en la práctica. Debemos tener cuidado de no elegir escenarios no representativos en este aspecto.

Ejemplo ¿Cómo usarías las características de comportamiento inherente para seleccionar un rango de tamaños de problema para el núcleo de resolución de ecuaciones?.

Respuesta Para este núcleo, el que exista trabajo suficiente y una distribución equilibrada de la carga dictaría que tuviésemos particiones de al menos 32×32 puntos. Para una máquina con 64 (8×8) procesadores, esto significa que el tamaño de la malla total sea de, al menos, 256×256. Este tamaño de malla requiere la comunicación de 4×32, ó 128, puntos de la malla por proceso en cada iteración para un cálculo de 32×32 ó 1K puntos. A cinco operaciones en punto flotante y 8 bytes por punto en la malla da lugar a un ratio de comunicación/cálculo inherente de 1 byte por cada cinco operaciones en punto flotante. Suponiendo un procesador puede realizar este cálculo a 200 MFLOPS, obtenemos un requisitos en ancho de banda de 40 MB/s. Este valor es bastante pequeño para las redes de los modernos multiprocesadores. Supongamos que un valor por debajo de 5 MB/s en la comunicación es asintóticamente bajo para nuestro sistema. Desde el punto de vista de las propiedades inherentes, no hay necesidad de ejecutar problemas mayores de 256×256 puntos (únicamente $64 \times 8B$ ó 521KB de datos) por proceso, o un total de $2K \times 2K$ mallas.

Las características inherentes como la distribución de la carga y comunicación varían suavemente con el tamaño del problema, por lo tanto, para tratar con estas características podemos coger unos pocos tamaños que abarquen el espectro de interés. Si los ratios varían muy lentamente, no necesitaremos elegir muchos tamaños. La experiencia muestra que el tres es un buen número en la mayoría de los casos. Por ejemplo, para el núcleo de resolución de ecuaciones podemos elegir mallas de 256×256, $1K \times 1K$ y $2K \times 2K$.

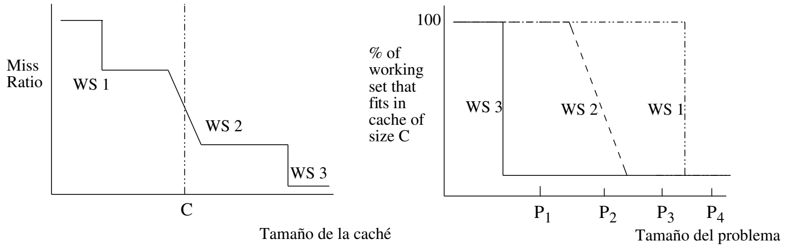
Por otra parte, las interacciones de la localidad temporal y espacial con la arquitectura exhiben umbrales en sus efectos en el rendimiento, incluyendo la generación de comunicaciones artificiosas cuando cambia el tamaño del problema. Podemos necesitar extender nuestra elección de tamaños de problema lo bastante para cubrir estos umbrales. Al mismo tiempo, la naturaleza del umbral nos puede ayudar a limitar el espacio de parámetros. El siguiente paso en la elección del tamaño de los problemas es examinar la localidad temporal y los conjuntos de trabajo.

Paso 3: Usar la localidad temporal y los conjuntos de trabajo

El que los conjuntos de trabajo quepan o no en la caché local puede afectar dramáticamente a las características de ejecución, como el tráfico en la memoria local y la comunicación artificiosa, incluso si la comunicación inherente y el balance de carga computacional no cambia mucho. En aplicaciones como Raytrace, los conjuntos de trabajo importantes son grandes y consisten en datos que están asignados en su mayoría a nodos remotos, con lo que la comunicación artificiosa debida a limitaciones en la capacidad de replicación puede dominar las comunicaciones inherentes. Esta comunicación artificiosa tiende a crecer en lugar de disminuir cuando se incrementa el tamaño del problema. En otras aplicaciones (como Ocean), los conjuntos de trabajo que no se ajustan a la caché pueden generar mucho más tráfico a la memoria local en lugar de comunicación artificiosa. Debemos incluir tamaños de problema que representen ambos lados del umbral (que se ajusten o no) para los conjuntos de trabajo importantes, si tales tamaños de problema son reales en la práctica. De hecho, cuando sean realistas para la aplicación, debemos incluir además un tamaño de problema que sea muy grande para la máquina, por ejemplo, uno que casi llene la memoria, incluso aunque este tamaño de problema pueda no ser interesante desde el punto de vista del equilibrio de la carga y comunicación inherente. Los problemas grandes a menudo ejercitan las interacciones entre la arquitectura y el sistema operativo que no ocurren para problemas más pequeños, como fallos en la TLB, fallos de página, y una gran cantidad de tráfico debido a fallos de capacidad en las cachés. Los siguientes ejemplos nos ayudarán a ilustrar como podemos elegir el tamaño del problema en función de los conjuntos de trabajo.

Ejemplo Supongamos que una aplicación presenta una curva de la tasa de fallo con respecto al tamaño de caché como la que se muestra en la figura 5.5(a) para un tamaño de problema y número de procesadores fijo y para la caché de más bajo nivel de un nodo de nuestra máquina (e.d., la caché que está más lejana al procesador y más cercana a la memoria). Si *C* es el tamaño de la caché, ¿cómo influye esta curva en la elección del tamaño de los problemas usados para evaluar la máquina?.

Respuesta Podemos ver a partir de la figura 5.5(a) que para el tamaño del problema (y el número de procesadores) mostrado, el primer conjunto de trabajo se ajusta en la caché de tamaño *C*, el segundo parcialmente, y el tercero no cabe en la caché. Cada uno de estos conjuntos de trabajo varía con el tamaño del problema de manera diferente. Este escalamiento determina a qué tamaño del problema ese conjunto de trabajo puede dejar de caber en una caché de tamaño *C* y, por tanto, qué tamaños de problema debemos elegir para cubrir los casos representativos. De hecho, si la curva consiste realmente en acodamientos abruptos, entonces podemos dibujar un tipo de curva diferente, en este caso una por cada conjunto de trabajo de importancia. Esta curva, mostrada en la figura 5.5(b), representa si un conjunto de trabajo cabe en nuestra caché de tamaño *C* cuando cambia el tamaño del problema. Si el tamaño del problema para el cual ocurre un acodamiento en esta curva está dentro del rango de tamaños de problema que hemos determinado que son realistas, entonces debemos asegurarnos que incluimos un tamaño de problema de cada lado de ese acodamiento. No hacer esto puede provocar que nos perdamos importantes efectos cuando la arquitectura de la memoria o de la comunicación está sometida a una gran utilización. El hecho de que las curvas sean planas a ambos lados de un acodamiento en este ejemplo significa que todo lo que debemos preocuparnos acerca de la caché es la tasa de fallos, por lo tanto sólo necesitamos elegir un tamaño de problema en cada lado del acodamiento para este propósito y eliminar el resto[1].



(a) (b)

Figura 5.5: Elección del tamaño del problema en función del los conjuntos de trabajo que quepan en la caché.

Ejemplo ¿Cómo influyen los conjuntos de trabajo en nuestra elección del tamaño de los problemas para el núcleo de resolución de ecuaciones?.

Respuesta Los conjuntos de trabajo más importantes para el núcleo de resolución de ecuaciones se encuentra cuando dos subfilas de una partición caben en la caché y cuando la partición completa de un procesador cabe en la caché. Ambas casos están definidos muy acusadamente en este núcleo. Incluso con la malla más grande que elijamos basándonos en el ratio inherente de comunicación/cálculo ($2K \times 2K$), el conjunto de trabajo por procesador es únicamente de 0.5 MB, por lo que ambos conjuntos de trabajo caben en la caché confortablemente (si se utiliza una representación en array 4D, no se producen fallos debidos a conflictos). Así, podemos necesitar elegir también tamaños de problema más grandes. Para que el primer conjunto de trabajo de dos subfilas exceda una caché de 1 MB necesitamos una subfila de 64 K puntos, lo que da una malla total de $64K \times \sqrt{64}$ o 512 K filas (o columnas) para nuestra máquina de 64 procesadores. Esto es un conjunto de datos de 32 GB por procesador, que es demasiado grande para ser realista. Sin embargo, es realista suponer que el otro conjunto de trabajo importante —la partición de un procesador— no cabe en una caché de 1 MB. Podemos conseguir esto eligiendo un tamaño del problema de, digamos, 512×512 puntos (2 MB) por procesador o un total de $4K \times 4K$. Este tamaño no termina de llenar la memoria de la máquina, por lo que podemos elegir un tamaño de problema más para este propósito, digamos, un total $16K \times 16K$ puntos o 32 MB por procesador. Ahora tenemos cinco tamaños de problema: 256×256, $1K \times 1K$, $2K \times 2K$, $4K \times 4K$ y $16K \times 16K$.

Paso 4: Usar la localidad espacial

Supongamos que la estructura de datos usada para representar la malla en una implementación de espacio de direcciones compartida del núcleo de resolución de ecuaciones es un array bidimensional. Una partición para un procesador, que es uno de los conjuntos de trabajo importantes, puede no mantenerse en su caché a lo largo de las iteraciones sobre la malla. Incluso si la capacidad de la caché

---

[1] La reducción de una región plana de la curva de tasa de fallo no es necesariamente apropiado si estamos preocupados por aspectos del comportamiento de la caché diferentes de la tasa de fallo y que también se ven afectados por el tamaño de la caché.

es suficiente, los conflictos en la caché pueden ser bastante frecuentes ya que las subfilas de la partición de un procesador no son contiguas en el espacio de direcciones. En cualquier caso, si el conjunto de trabajo no cabe, entonces es importante que la partición de un procesador se almacene en su memoria local en una máquina de memoria distribuida. La granularidad de almacenamiento en la memoria principal es una página, con un tamaño típico de 4-16 KB. Si el tamaño de una subfila es menor que el tamaño de la página, el almacenamiento apropiado se hace difícil y puede llegar a producirse una gran cantidad de comunicación artificiosa. Sin embargo, si una subfila es un múltiplo del tamaño de la página, el almacenamiento no presenta problemas y existe poca comunicación artificiosa. Ambos escenarios pueden ser realistas, por lo que debemos representar ambos. Si el tamaño de la página es de 4 KB, los tres primeros tamaños del problema que hemos elegido presentan un tamaño de subfila menor de 4 KB, por lo que no podemos distribuirlos apropiadamente; los dos últimos tienen subfilas mayores o iguales a 4 KB, por lo que podemos realizar una buena distribución supuesto que la malla está alineada en frontera de página. Así, no es necesario expandir nuestro conjunto de tamaños de problema. Con una representación de la malla en un array 4D, la partición de la malla para un proceso es continua en el espacio de direcciones, por lo que la distribución de datos es sencilla siempre que las particiones sean lo bastante grandes para que ésta sea necesaria.

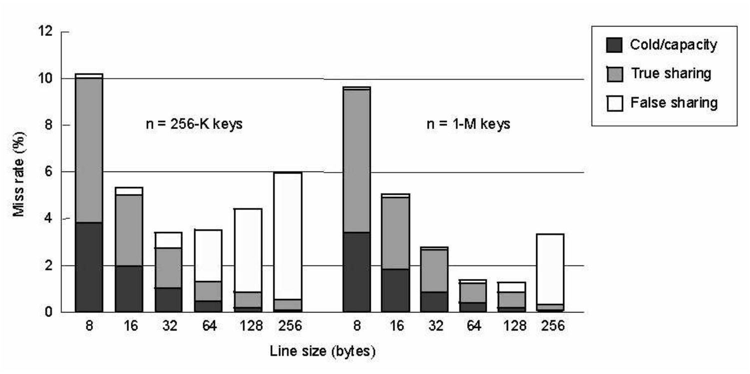


Figura 5.6: Impacto del tamaño del problema y número de procesadores en el comportamiento de la localidad espacial para la ordenación Radix.

Un ejemplo más completo de las interacciones de la localidad espacial y de sus interacciones en la arquitectura podemos encontrarlo en el programa de ordenación Radix, que describiremos posteriormente en este capítulo. La figura 5.6 muestra, para este problema, como cambia la tasa de fallo en función del tamaño del bloque y para dos tamaños de problema n diferentes (ordenación de 256 K y 1 M enteros). Los datos se han obtenido ejecutando el programa sobre una máquina de espacio de direcciones compartido con coherencia de cachés y manteniendo el número de procesador p. El componente de falsa compartición de la tasa de fallo tiende a incrementarse con el tamaño de bloque de la caché. Cuando esta tasa se hace significativa, da lugar a mucha comunicación artificiosa y puede destruir el rendimiento de esta aplicación. Para un tamaño de bloque dado en nuestra máquina, la falsa compartición puede o no destruir el rendimiento de la ordenación dependiendo del tamaño del problema (compárense los resultados para un tamaño de bloque de 64 bytes). De aquí se deduce que para un tamaño de bloque dado, la falsa compartición es grande si el ratio entre el tamaño del problema y el número de procesadores es menor que un cierto umbral e insignificante si es mayor.

Muchas aplicaciones muestran estos efectos umbral en las interacciones de la localidad espacial con el tamaño del problema; en otros, especialmente en muchas aplicaciones irregulares, como Barnes-Hut y Raytrace, las estructuras de datos y patrones de acceso son tales que la localidad espacial no se incrementa con el tamaño del problema. Identificar la presencia de tales umbrales requiere el entendimiento de la localidad de las aplicaciones y sus interacciones con los parámetros de la arquitectura e ilustra algunas de las sutilezas existentes en la evaluación.

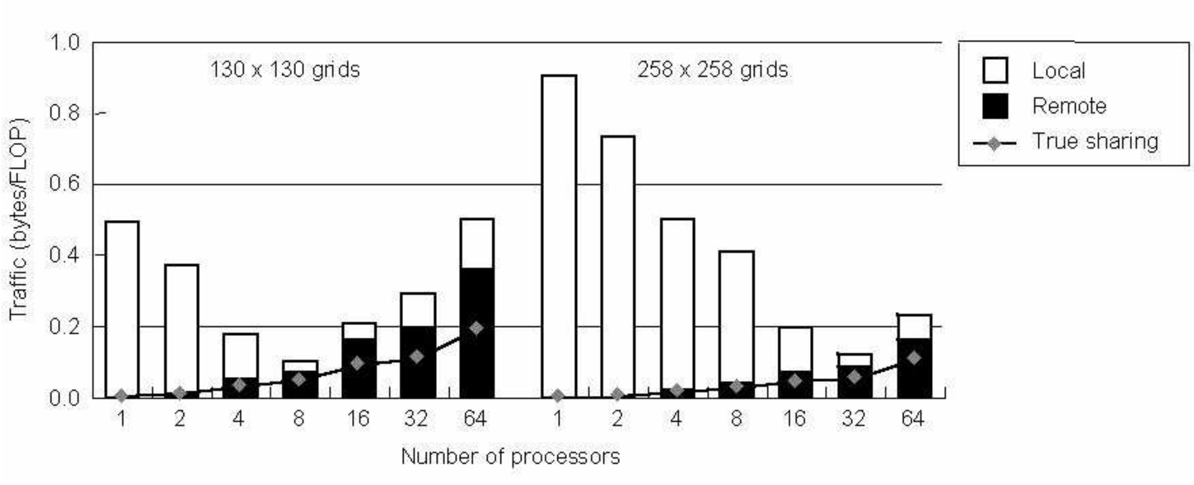


Figura 5.7: Efecto del tamaño del problema, número de procesadores y conjuntos de trabajo que quepan en la memoria.

En resumen, el núcleo de resolución de ecuaciones ilustra la dependencia de muchas características de ejecución del tamaño del problema, algunas exhiben acodamientos en umbrales que interactúan con los parámetros de la arquitectura y otros no. Con mallas de $n \times n$ y p procesadores, si el ratio $\frac{n}{p}$ es grande, entonces el ratio comunicación/cálculo es bajo, los conjuntos de trabajo importantes es poco probable que quepan en las cachés de los procesadores dando lugar a tasas de fallos por capacidad altas, pero la localidad espacial es buena incluso cuando se utiliza un representación de arrays bidimensionales. La situación es la opuesta cuando $\frac{n}{p}$ es pequeño: alto ratio comunicación/cálculo, pobre localidad espacial y falsa compartición (con la representación 2D), y pocos fallos de capacidad local. Así, el cuello de botella dominante en el rendimiento cambia del acceso local en un caso a la comunicación en el otro. La figura 5.7 ilustra estos efectos para la aplicación Ocean que tiene un núcleo similar al de la resolución de ecuaciones.

Otras aplicaciones pueden exhibir dependencias diferentes y específicas en función del tamaño del problema. Mientras que no existan fórmulas universales para elegir el tamaño de los problemas para evaluar una máquina, los pasos mostrados aquí suponen una metodología útil y debería asegurar que los resultados obtenidos para una máquina no se deben a artificios que pueden quitarse del programa. Si vamos a comparar dos máquinas, es útil elegir tamaños de problema que utilicen los escenarios anteriores sobre ambas máquinas. A pesar de la gran variedad de puntos a considerar, la experiencia muestra que el número de tamaños de problema necesarios para evaluar un máquina de tamaño fijo con un aplicación es normalmente bastante pequeño ya que sólo existen unos pocos umbrales importantes.

5.2.4 Variando el tamaño de la máquina

Supongamos que queremos evaluar el rendimiento de la máquina cuando cambia el número de procesadores. Ya hemos visto como varía el tamaño del problema bajo diferentes modelos de escalamiento y qué métricas se usan para el incremento del rendimiento debido al paralelismo. El punto que queda es cómo se elige el tamaño del problema, para un tamaño de máquina, como un punto de partida desde el cual escalar. Una estrategia es comenzar con los tamaños de problema que elegimos previamente para un número fijo de procesadores y escalar cada uno de ellos de acuerdo con los diferentes modelos de escalamiento. Podemos simplificar nuestro rango de tamaños de problema base a tres —uno pequeño, uno mediano y uno grande— que con tres modelos de escalamiento daría lugar a nueve conjuntos de datos de rendimiento y curvas de speedup. Sin embargo, esto requeriría asegurarnos que los tamaños de los problemas, cuando se reducen, enfatizan las capacidades de las máquinas más pequeñas.

Una estrategia alternativa es comenzar con unos pocos tamaños de problema bien elegidos sobre un uniprocesador y escalarlos bajo los tres modelos. En este caso, también es razonable elegir tres tamaños de problema para un uniprocesador. El problema pequeño debería ser tal que su conjunto de trabajo coja en la caché de un uniprocesador. Este problema no será muy útil bajo el escalamiento de tamaño fijo sobre grandes máquinas pero funcionará bajo el escalamiento restringido por la memoria e incluso por el tiempo. El problema grande debería ser tal que su conjunto de trabajo importante no coja en la caché de un uniprocesador. Bajo el escalamiento de tamaño fijo el conjunto de trabajo puede caber en algún punto (si se reduce con un creciente número de procesadores), mientras que bajo el escalamiento de memoria fija es menos probable que se ajuste a la caché y es probable que siga generando tráfico de capacidad. Una elección razonable para un problema grande es aquel que quepa en su mayoría en la memoria de un único nodo o tarde mucho tiempo en ejecutarse sobre él. De esta manera, seguirá llenando la memoria incluso sobre grandes sistemas bajo el modelo de escalamiento limitado por memoria. El problema de tamaño medio puede elegirse de una manera juiciosa entre ambos tamaños; si es posible, debe tardar un tiempo sustancial en un uniprocesador. El punto sobresaliente es cómo explorar el escalamiento bajo el modelo de tamaño fijo para tamaños de problema que no quepan en la memoria de un único nodo sin experimentar problemas de speedup superlineales. En este caso la solución es elegir simplemente un tamaño de problema y una medición del speedup que no sea relativa a un único procesador sino al número de procesadores para el cual el problema cabe en la memoria.

5.2.5 Elección de las métricas de rendimiento

Una pregunta importante en la evaluación o comparación de máquinas es especificar las métricas que deben usarse. Al igual que es fácil llegar a conclusiones erróneas al no elegir apropiadamente las cargas de trabajo y los parámetros; también es fácil llevarse una impresión errónea cuando no se mide y presentan los resultados de forma entendible. En general, tanto el coste como el rendimiento son métricas importantes para comparar máquinas o evaluar rendimiento. Y en la evaluación de cómo de bueno es el escalamiento de una máquina cuando se añaden recursos (por ejemplo, procesadores y memoria), no es únicamente cómo se incrementa el rendimiento lo que importa sino también cómo se incrementa el coste. Incluso si el rendimiento aumenta mucho menos que linealmente, si el coste de los recursos necesarios para ejecutar el programa no se incrementa mucho más rápidamente que el rendimiento, entonces puede ser efectivo en cuanto a coste el utilizar una máquina mayor. De forma global, algunas medidas de “coste-rendimiento” son más apropiadas que el rendimiento por sí solo.

Sin embargo, el coste y el rendimiento pueden medirse por separado, y el coste es muy dependiente del mercado. Nos centraremos, por tanto, en la métricas para medir el rendimiento.

El rendimiento absoluto y la mejora de rendimiento debido al paralelismo son dos métricas útiles. En este punto examinaremos los puntos más delicados en el uso de estas métricas para evaluar y, especialmente, comparar máquinas y consideraremos el papel de otras métricas que se basan en el ratio de procesamiento (p.e., megaflops), utilización de recursos y tamaño del problema en lugar utilizar directamente el trabajo y el tiempo. Algunas métricas son claramente importantes y deben de estar siempre presentes, mientras que la utilidad de otras depende de lo que busquemos y del entorno en el que operamos.

Rendimiento absoluto

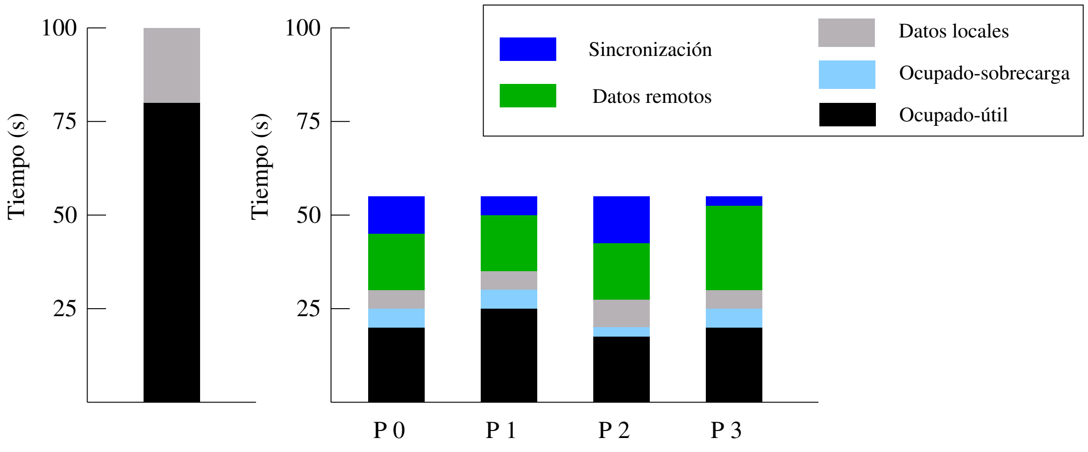
Para un usuario de un sistema el rendimiento absoluto es la métrica de rendimiento que más le importa. Supongamos que el tiempo de ejecución es nuestra métrica para el rendimiento absoluto. El tiempo puede medirse de varias formas. En primer lugar, existe una elección entre el tiempo de usuario y el tiempo de reloj de pared para una carga de trabajo. El *tiempo de usuario* es el tiempo que la máquina está ejecutando la carga de trabajo, excluyendo la actividad del sistema y otros programas que puedan estar compartiendo la máquina; el *tiempo de reloj de pared* es el tiempo total que ha tardado la carga de trabajo —incluyendo toda la actividad que haya existido. En segundo lugar, está el tema de si usar el tiempo de ejecución medio o máximo de los procesos del programa.

Dado que a los usuarios les importa el tiempo de reloj, debemos medir y mostrar esta medida cuando se comparan sistemas. Sin embargo, si otros programas de usuario —no únicamente el sistema operativo— interfieren con la ejecución del programa como resultado de la multiprogramación, el tiempo de reloj no nos ayuda a entender los cuellos de botella del rendimiento. Obsérvese que el tiempo de usuario para ese programa tampoco es muy útil en ese caso, ya que la ejecución solapada con procesos no relacionados corrompe las interacciones del programa en el sistema de memoria así como su comportamiento en la sincronización y la distribución de la carga. Debemos por lo tanto presentar siempre el tiempo de reloj y describir el entorno de ejecución (por lotes o multiprogramado), independientemente de si presentamos una información más detallada conducente a un mejor entendimiento. Y si queremos entender el rendimiento de una aplicación concreta, debemos ejecutarla aisladamente con, tal vez, la única intervención del sistema operativo.

De forma similar, ya que un programa paralelo no finaliza hasta que el último proceso a finalizado, es el tiempo de este instante el que es importante, no la media de los procesos. Las medias tienden a no enfatizar la falta de una distribución equilibrada. Por supuesto, si queremos entender realmente los cuellos de botella del rendimiento, deberíamos ver los perfiles de ejecución de todos los procesos —o al menos una muestra— indicando la descomposición de los diferentes componentes del tiempo (ver figura 5.8). Los componentes del tiempo de ejecución nos dicen por qué un sistema mejora a otro y si la carga de trabajo es apropiada para la investigación (e.d., no está limitada por desequilibrios en la carga).

Mejora del rendimiento o speedup

Una pregunta que surge al medir el speedup para cualquier modelo de escalamiento es qué debe medir realmente el denominador en la expresión del speedup —el rendimiento sobre un procesador. Existen cuatro opciones:



(a) Secuencial (b) Paralelo con cuatro procesadores

Figura 5.8: Componentes del tiempo de ejecución desde la perspectiva de un procesador individual.

1. El rendimiento del programa paralelo sobre un procesador de la máquina paralela.
2. El rendimiento de una implementación secuencial del mismo algoritmo sobre un procesador de la máquina paralela.
3. El rendimiento del "mejor" algoritmo secuencial y programa para el mismo problema sobre un procesador de la máquina paralela.
4. El rendimiento del "mejor" programa secuencial en una máquina estándar de consenso.

La diferencia entre (1) y (2) es que el programa paralelo incurre en sobrecargas incluso cuando se ejecuta sobre un uniprocesador, dado que ejecuta operaciones de sincronización, instrucciones de manejo del paralelismo, código de particionamiento, o incluso los tests para omitir éstos. Esta sobrecarga puede ser a veces significativa. La distinción entre (2) y (3) es que el mejor algoritmo secuencial puede no ser fácilmente paralelizable de forma efectiva; por tanto, el algoritmo usado en el programa paralelo puede ser diferente del mejor algoritmo secuencial.

El uso del rendimiento definido en (3) da lugar claramente a una métrica del rendimiento mejor y más precisa que en el caso (1) y (2) desde la perspectiva del usuario. Desde un punto de vista arquitectónico, sin embargo, puede ser correcto utilizar la definición (2). La definición (4) no tiene en cuenta el rendimiento del uniprocesador de la máquina y los resultados en una comparación de métricas son similares a los del rendimiento absoluto.

Ratio de procesamiento

Una métrica que se cita a menudo para caracterizar el rendimiento de máquinas es el número de operaciones de ordenador que ejecutan por unidad de tiempo (en oposición a las operaciones que tienen significado a nivel de aplicación, como las transacciones o enlaces químicos). Los ejemplos clásicos son MFLOPS y MIPS. Estas métricas no son, en general, buenas métricas para medir el rendimiento a pesar de que son bastante populares entre los vendedores de ordenadores. La razón básica es que, al menos que utilicemos una medida no ambigua e independiente de la máquina del número de FLOPs o instrucciones necesarias para resolver un problema, en lugar del número realmente utilizado, estas medidas se pueden inflar artificialmente.

Utilización

A veces los arquitectos miden el éxito de un diseño determinado el porcentaje de tiempo que son capaces de mantener sus procesadores ocupados ejecutando instrucciones en lugar de que estén parados esperando un resultado. Debería quedar claro, sin embargo, que la utilización del procesador no es una métrica de interés para un usuario y que no es una buena métrica del rendimiento por sí sola. Ésta puede inflarse artificialmente y no nos dice mucho acerca del rendimiento final o de los cuellos de botella. Sin embargo, puede ser útil como punto de comienzo para decidir si es necesario un examen más profundo para detectar problemas de rendimiento en un programa o máquina. Los argumentos aquí expuestos también son válidos para la utilización de otros recursos; la utilización es útil para determinar si el diseño de una máquina está equilibrado y donde están los cuellos de botella, pero no es útil para la medida y para comparar rendimientos.

El tamaño del problema

Otra métrica interesante es el problema de menor tamaño para una aplicación dada que obtiene una *eficiencia paralela* dada, que se define como el speedup dividido por el número de procesadores (bajo un modelo de escalamiento dado). Dado que, normalmente, las sobrecargas debidas al paralelismo decrecen con el tamaño del problema, el beneficio de una mejora en la arquitectura de las comunicaciones puede verse a menudo en la posibilidad de ejecutar problemas más pequeños con un buen comportamiento. Mantener la eficiencia paralela fija cuando aumenta el número de procesadores introduce, en algún sentido, un nuevo modelo de escalamiento que podríamos llamar *escalamiento limitado por la eficiencia*. Por supuesto, esta métrica debe utilizarse cuidadosamente ya que los efectos de capacidad pueden dominar las diferencias en las comunicaciones y los problemas pequeños pueden fallar en enfatizar aspectos importantes del sistema. La eficiencia paralela es útil, pero no una métrica de rendimiento general.

Mejora en porcentaje del rendimiento

Una métrica que se usa a veces para evaluar la mejora en el rendimiento debido a una característica de la arquitectura es la mejora en porcentaje del tiempo de ejecución o speedup proporcionado por la característica. Sin mencionar el rendimiento paralelo original (e.d. el speedup original), esta métrica puede dar lugar a equívocos en sistemas paralelos. Por ejemplo, mejorar la aceleración de 400 a 800 en un sistema de 1024 procesadores tiene el mismo porcentaje que mejorar la aceleración de 1.1 a 2.2, pero el último caso carece de interés para un sistema con 1024 procesadores. Si el tamaño del problema es la razón para un bajo speedup, se da a menudo el caso que incrementar el tamaño del problema para obtener speedup decentes reduce frecuentemente de forma drástica la mejora conseguida por la característica. También en este caso la métrica tiene valor, pero debe suministrarse con otras métricas para evitar equívocos.

En resumen, tanto el coste como el rendimiento son métricas importantes a considerar. Desde el punto de vista de usuarios que comparan máquinas, la métrica de rendimiento de mayor interés es el tiempo de ejecución real. Sin embargo, desde el punto de vista de un arquitecto, o de un programador que intenta entender el rendimiento de un programa, es mejor ver tanto el tiempo de ejecución como el speedup. Ambas métricas deben de estar presentes en los resultados de cualquier estudio. Idealmente, el tiempo de ejecución debe descomponerse en sus componentes principales. Para entender los cuellos de botella en el rendimiento, es muy útil obtener esta descomposición de los distintos componentes del tiempo de ejecución por proceso o como media y una medida de dispersión de todos los procesos (la media por sí sola no es suficiente). En la evaluación de cambios en la arquitectura de las comunicaciones, o al comparar máquinas paralelas basadas en nodos equivalentes, pueden ser útiles métricas de tamaño o configuración base como el mínimo tamaño de problema necesario para conseguir un cierto objetivo. Las métricas como MFLOPS, MIPS y utilización del procesador se pueden utilizar para propósitos específicos, pero para usarlas para representar el rendimiento es necesario realizar muchas suposiciones sobre el conocimiento y la integridad de quien las presenta, debido al historial de mala utilización que existe sobre estas métricas.

5.3 Evaluación de una idea arquitectónica o decisión de compromiso

Imagina que eres un arquitecto de una compañía de ordenadores, listo para diseñar el multiprocesador de la siguiente generación. Supongamos que tienes una nueva idea sobre la arquitectura y te gustaría decidir si incluirla o no en la máquina. Puedes tener una abundante información acerca del rendimiento y los cuellos de botella de la máquina de la generación anterior, lo que incluso puede haber sido la causa primera por la que te ha surgido la idea. Sin embargo, la idea y los datos no es lo único nuevo. La tecnología ha cambiado desde la última máquina, desde el nivel de integración hasta los tamaños de las cachés y las organizaciones usadas por los microprocesadores. El procesador que vas a utilizar puede ser no únicamente mucho más rápido sino mucho más sofisticado (p.e., superescalar de 4 vías y manejo dinámico de instrucciones frente a una arquitectura escalar y manejo estático de instrucciones), y puede tener nuevas capacidades que afecten a la idea. Es probable que el sistema operativo haya cambiado también, como el propio compilador e incluso las cargas de trabajo de interés, y estos componentes software pueden cambiar aún más para cuando la máquina se haya construido y se esté vendiendo realmente. La característica en la que estás interesado tiene un coste significativo en hardware y sobre todo en tiempo de diseño. Y existen fechas topes que tienes que cumplir.

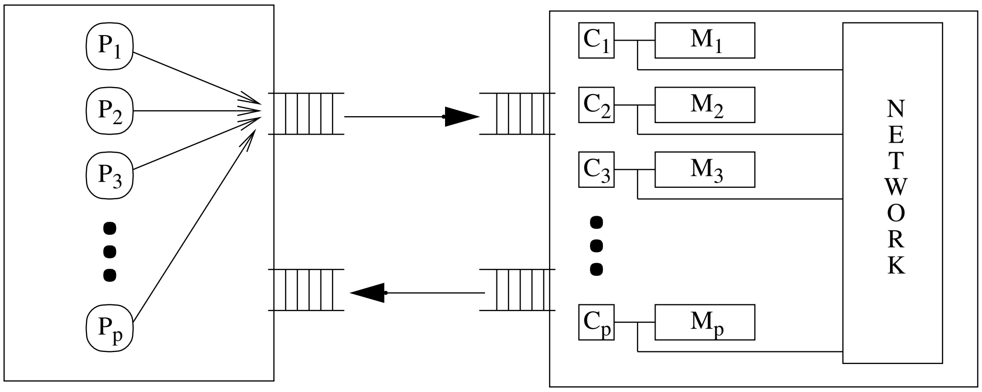
En este mar de cambios, la relevancia de los datos que tienes para tomar decisiones acerca del rendimiento y coste es cuestionable. En esta situación y, sobre todo, si la característica presenta un coste alto, sería deseable construir un simulador que modele tu sistema. Fijando el resto de parámetros — compilador, sistema operativo, procesador, parámetros tecnológicos y de arquitectura — a las configuraciones estimadas y simulando el sistema con y sin la característica de interés, es posible juzgar su impacto en el rendimiento.

En esta sección discutiremos las consideraciones metodológicas en la elección de parámetros y acotación del espacio de diseño para estudios mediante simulación, usando una evaluación concreta como ejemplo. En primer lugar, veamos como se realiza la simulación de un multiprocesador.

5.3.1 Simulación multiprocesador

Aunque se simulen múltiples procesos y nodos de procesamiento, la simulación puede ejecutarse sobre un único procesador. Un *generador de referencias* juega el papel de los procesadores en la máquina paralela. Simula las actividades de los procesadores y las referencias emitidas a la memoria (junto con un identificador de proceso que indica de qué procesador es la referencia) u órdenes (como envíos y recepciones) a un simulador del sistema de memoria y red de interconexión (ver figura 5.9). Si la simulación se está ejecutando en un uniprocesador, los diferentes procesos simulados comparten el tiempo del uniprocesador, planificados por el generador de referencias. Un ejemplo de planificación sería quitar del planificador a un proceso cada vez que envía una referencia al simulador del sistema de memoria y permitir que otro proceso se ejecute hasta que ese proceso genere su siguiente referencia; otro ejemplo sería replanificar los procesos en cada ciclo de reloj simulado. El simulador del sistema de memoria simula todas las cachés y las memorias principales de los diferentes nodos de proceso, al igual que la propia red de interconexión.

El acoplamiento entre el generador de referencias (simulador de procesadores) y el simulador del sistema de memoria puede organizarse de varias maneras, dependiendo de la exactitud que se necesite en la simulación y la complejidad del modelo de procesador. Una opción es una *simulación dirigida por traza*. En este caso, se obtiene en primer lugar una traza de las instrucciones ejecutadas por cada proceso de un programa paralelo. Esta traza reemplaza al generador de referencias: las instrucciones de la traza alimentan al simulador que simula la jerarquía de memoria extendida del multiprocesador de interés. En este caso, el acoplamiento o flujo de información es sólo en una dirección: del generador de referencias (aquí únicamente una traza) al simulador del sistema de memoria.



Generador de referencias

Simulador de la memoria y de la red de interconexión

Figura 5.9: Simulación multiprocesador dirigida por la ejecución.

Una forma más popular de simulación es la *simulación dirigida por ejecución*, que proporciona un acoplamiento en ambas direcciones. En la simulación dirigida por ejecución, cuando el simulador del sistema de memoria recibe una referencia u orden del generador de referencias (que es ahora un programa en lugar de una traza predeterminada), simula el camino de la referencia a través de la jerarquía de memoria extendida —incluyendo la contención con el resto de referencias— y devuelve al generador de referencias el tiempo en que la referencia se satisfará. Esta información, junto con aspectos concernientes con la equidad y la preservación de la semántica de los eventos de sincronización, se usa en el generador de referencias para determinar qué proceso simulado se planificará a continuación y cuándo emitir la siguiente instrucción de ese proceso. Así, tiene lugar una retroalimentación desde el simulador del sistema de memoria al generador de referencias, influenciando la actividad del último como en una máquina real y proporciona una mayor precisión que en la simulación basada en trazas. Para permitir una máxima concurrencia en los eventos de simulación y referencias, la mayoría de los componentes del sistema de memoria y de la red se modelan también como hilos de comunicación separados que se planifican mediante el simulador. Una notación global del *tiempo simulado* —es decir, el tiempo virtual que sería visto por la máquina simulada, no el tiempo real de ejecución del propio simulador— se mantiene por el simulador. Este es el tiempo que miraremos a la hora de determinar el rendimiento de las cargas de trabajo en la arquitectura simulada y que se usa para realizar las decisiones de planificación. Además del tiempo, los simuladores mantienen unas extensas estadísticas acerca de la ocurrencia de distintos eventos de interés. Esto proporciona una abundancia de información detallada sobre el rendimiento que sería difícil, si no imposible, de obtener en un sistema real. Sin embargo, los resultados pueden adolecer de falta de credibilidad ya que es, después de todo, una simulación.

5.3.2 Reducción del problema y de los parámetros de la máquina para la simulación

Dado que la simulación se realiza por software e incluye a muchos procesos o hilos que deben ser planificados muy frecuentemente (a mayor frecuencia mayor exactitud), no es sorprendente que la simulación sea muy costosa. Actualmente, se están realizando investigaciones en la área de la simulación para que ésta sea más rápida y usar emulación por hardware en lugar de la simulación, pero el progreso no ha sido suficiente para cambiar el hecho de que los parámetros deben reducirse sustancialmente.

La parte con enjundia en la reducción del problema y de los parámetros de la máquina es que queremos que la máquina reducida ejecutando el problema más pequeño sea representativa de la máquina completa ejecutando el problema más grande. Desafortunadamente, no existen buenas fórmulas para hacer esto. No obstante, es un tema importante ya que es la realidad con la que se encuentran la mayoría de las evaluaciones de decisiones de compromiso en la arquitectura. Por lo tanto debemos, al menos, entender las limitaciones de tal escalamiento, reconociendo qué parámetros pueden reducirse con confianza y cuáles no, y desarrollar guías que nos ayuden a evitar la mayoría de los fallos. Examinemos en primer lugar la reducción del tamaño del problema y del número de procesadores para explicar posteriormente algunas dificultades posteriores asociadas con los parámetros de bajo nivel de la máquina. De nuevo nos centraremos en una abstracción de comunicación de un espacio de direcciones compartido con coherencia de cachés.

Parámetros del problema y número de procesadores

Consideremos en primer lugar los parámetros del problema. En primer lugar debemos buscar aquellos parámetros del problema, si existen, que afecten al tiempo de simulación de forma notable pero que tengan poco impacto en las características de la ejecución relacionadas con el rendimiento paralelo. Un ejemplo es el número de pasos de tiempo en muchas aplicaciones científicas, como Ocean o incluso Barnes-Hut, o el número de iteraciones en la resolución de ecuaciones. Los valores de los datos manipulados pueden cambiar mucho a lo largo del tiempo, pero las características de comportamiento no cambian mucho. En estos casos, podemos ejecutar la simulación durante unos pocos pasos de tiempo[2].

---

[2] Por supuesto, debemos omitir los periodos de inicialización y comienzo en frío de la aplicación de las medidas ya que su impacto es mucho mayor con un número de pasos de tiempo reducido de lo que sería en la práctica. Si esperamos que el comportamiento sobre largos periodos de tiempo puedan cambiar sustancialmente, como es posible en Barnes-Hut o en aplicaciones cuyas caractarísticas cambian más dinámicamente, podemos volcar el estado del programa de forma periódica de una ejecución sobre una máquina real de la configuración de problema que estamos simulando, y realizar algunas simulaciones de muestro con estos estados como datos de entrada.

Desafortunadamente, muchos parámetros de la aplicación afectan a las características de la ejecución relacionadas con el paralelismo. Cuando se escalan estos parámetros, debemos reducir también el número de procesadores ya que en caso contrario podemos obtener características de comportamiento muy poco representativas. Sin embargo, esto es difícil de realizar de forma representativa ya que nos enfrentamos con muchas restricciones que son difíciles de satisfacer de forma individual y que pueden ser imposibles de reconciliar mutuamente. Estas incluyen las siguientes:

- *Preservar la distribución del tiempo en las distintas fases del programa.* Las cantidades relativas de tiempo utilizado realizando los diferentes tipos de cálculos —por ejemplo, en las fases de la construcción del árbol y el cálculo de las fuerzas de Barnes-Hut— cambiarán muy probablemente con el tamaño del problema y la máquina.
- *Preservar las características de comportamiento claves.* Estas incluyen los ratios comunicaciones/cálculos, distribución de carga, localidad temporal y espacial, que pueden escalar de diferentes maneras.
- *Preservar las relaciones de escalamiento entre los parámetros de la aplicación.*
- *Preservar los patrones de contención y comunicación.* Esto es particularmente difícil ya que, por ejemplo, la actividad intensiva es difícil de predecir o controlar.

Más que preservar la representatividad verdadera cuando se reduce, unas metas más realistas son cubrir, al menos, un rango de puntos de operación realistas con respecto a las características de comportamiento que nos importan más para nuestro estudio y evitar escenarios irreales. Así, las simulaciones reducidas no son cuantitativamente representativas pero pueden usarse para obtener perspectiva y estimaciones aproximadas. Con esta meta más modesta, supongamos que hemos reducido los parámetros de la aplicación y el número de procesadores de manera razonable y veamos como se escalan el resto de parámetros de la máquina.

Otros parámetros de la máquina

El tamaño del problema reducido y de la máquina interactúan de forma diferente con los parámetros de bajo nivel de lo que lo haría con los problemas completos. Por lo tanto, también tenemos que escalar estos parámetros de forma cuidadosa.

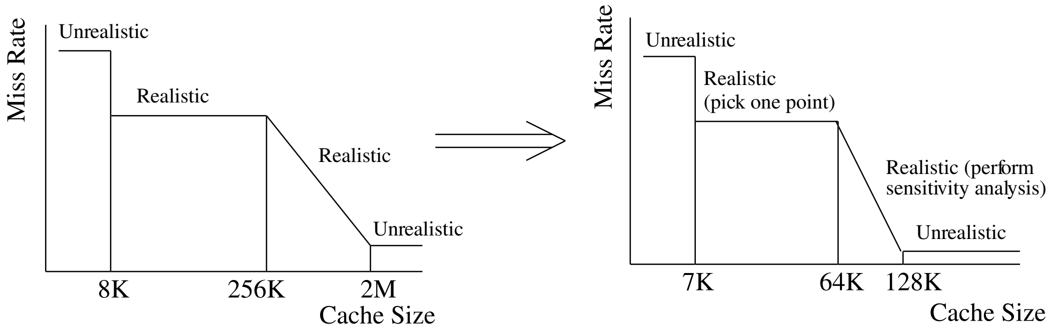
Consideremos el tamaño de la caché. Supongamos que el mayor problema y la configuración de la máquina que podemos simular para el núcleo de resolución de ecuaciones es una malla de 512×512 con 16 procesadores (p.e., 128 KB por procesador). Si no rebajamos el 1 MB de caché por procesador, nunca podremos representar la situación en donde el conjunto de trabajo importante no cabe en la caché. El punto clave con respecto al escalamiento de la cachés es que debe realizarse basándose en un conocimiento de como varían los conjuntos de trabajo relevantes según nuestra discusión de puntos de operación realistas e irreales (figura 5.5). No escalar el tamaño de la caché, o simplemente reducir su tamaño de forma proporcional al tamaño del conjunto de datos o tamaño del problema, es generalmente inapropiado ya que el tamaño de la caché interactúa más estrechamente con el tamaño del conjunto de trabajo, no con el conjunto de datos o el tamaño del problema. El siguiente ejemplo y la figura 5.10 ilustra cómo elegir el tamaño de las cachés dado un tamaño del problema y de la máquina. También debemos asegurarnos que las cachés que simulamos no se hacen demasiado pequeñas, ya que puede dar lugar a un mapeo no representativo y fragmentación artificiosa. También se aplican argumentos similares al resto del almacenamiento, incluyendo aquel que almacenan los datos que se comunican.

Ejemplo En la aplicación Barnes-Hut, supongamos que el tamaño del conjunto de trabajo más importante cuando se ejecuta el problema con $n = 1M$ partículas es $150KB$, si la máquina bajo estudio tiene cachés de $1MB$ por procesador, y sólo puedes simular una ejecución con $n = 16K$ partículas. ¿Sería apropiado reducir el tamaño de las cachés proporcionalmente al tamaño del conjunto de datos?. ¿Cómo elegirías el tamaño de la caché?.

Respuesta Como veremos posteriormente, el tamaño de los conjuntos de trabajo más importantes de Barnes-Hut varía en función de $\log n$, donde n es el número de partículas y es proporcional al tamaño del conjunto de datos. El conjunto de trabajo de $150KB$ cabe sin problemas en la caché de $1MB$ de la máquina original. Dada su lenta tasa de crecimiento, este conjunto de trabajo es probable que siempre quepa en la caché para problemas reales. Si escalamos el tamaño de la caché de forma proporcional al conjunto de datos para nuestra simulación, obtendremos un tamaño de caché de $1MB \times 16K/1M$, o $16KB$. El tamaño del conjunto de trabajo para el problema reducido es

$$150KB \times \frac{\log 16K}{\log 1M}$$

o $70KB$, que no cabe en la caché reducida de $16KB$. Así, este tipo de escalamiento de la caché nos ha llevado a un punto de operación que no es representativo de la realidad. Dado que esperamos que el conjunto de trabajo quepa en la caché en la realidad, debemos elegir un tamaño de caché lo suficientemente grande para que siempre quepa este conjunto de trabajo.



Full-Scale Problem and Machine

Scaled Down Problem and Machine

Figura 5.10: Elección del tamaño de las cachés para el problema y máquina reducida.

Si nos movemos hacia parámetros de más bajo nivel de la jerarquía extendida de la memoria, el escalamiento de los mismos de forma representativa se hace más difícil. Por ejemplo, las interacciones con la asociatividad de las cachés son muy difíciles de predecir y, normalmente, lo mejor que podemos hacer es dejar la asociatividad como está. El principal peligro es mantener una caché de mapeo directo cuando el tamaño de las cachés se reduce mucho ya que esta situación es particularmente susceptible a conflictos de mapeo que no ocurrirían en la caché original. Las interacciones con otros parámetros de la organización de la arquitectura de la memoria y las comunicaciones —como la granularidad de la asignación de datos, transferencia y coherencia— son también complejos e impredecibles a menos que exista una localidad espacial casi perfecta, pero el mantenerlos fijos da lugar, en muchos casos, a situaciones no representativas. Finalmente, los parámetros de rendimiento como la latencia, ocupación y ancho de banda son también muy difíciles de reducir apropiadamente para preservar la representatividad cuando cambian las frecuencias y patrones de comunicaciones.

En resumen, la mejor aproximación a la simulación es intentar ejecutar tamaños de problema realistas hasta que sea posible. Cuando la reducción es necesaria, debemos prestar atención a las líneas guía y fallos que hemos discutido para asegurar que se cubren los tipos importantes de los puntos de operación, y debemos extrapolar con cuidado. Nuestra confianza en la reducción descansa en nuestro entendimiento de la aplicación. En general, el uso de escenarios reducidos es válido para entender si ciertas características arquitecturales van a ser beneficiosas o no, pero es peligroso utilizarlo para llegar a conclusiones precisas sobre las situaciones a tamaño real.

5.3.3 Manejo del espacio de parámetros: evaluación de un ejemplo

Consideremos ahora el problema del gran espacio de parámetros que se abre cuando intentamos evaluar una idea en un contexto general. Para mantener la discusión de forma concreta, examinaremos la evaluación real que podríamos realizar usando simulación. Supongamos de nuevo una máquina de espacio de direcciones compartido con coherencia de cachés y con memoria físicamente distribuida. El mecanismo por defecto para las comunicaciones es la comunicación implícita en los bloques de las cachés a través de las cargas y almacenamientos, pero queremos explorar la reducción de la sobrecarga de la comunicación y el retraso de la misma mediante la comunicación de mensajes mayores. Por lo tanto, desearíamos entender la utilidad de añadir a una arquitectura la posibilidad de enviar mensajes más grandes, denominada *transferencia de bloques*, qué programas que pueden utilizarla además de los mecanismos estándar de transferencia de bloques de cachés (es decir, mezclar los modelos de programación de espacio de direcciones compartido y paso de mensajes). Por ejemplo, en la resolución de ecuaciones un proceso podría enviar una subfila o subcolumna completa de la frontera a su proceso vecino en una única transferencia de bloque.

En la elección de las cargas de trabajo para dicha evaluación, debemos elegir al menos alguna con comunicaciones que se puedan estructurar bien en mensajes largos, como en la resolución de ecuaciones. El problema más difícil es cubrir el espacio de los parámetros. Nuestro objetivos son, por tanto:

1. *Evitar características de ejecución irreales*. Debemos evitar combinaciones de parámetros (o puntos de operación) que den lugar a características de comportamiento irreales —es decir, comportamientos que no se encontrarían en el uso práctico de la máquina.
2. *Obtener un buen cubrimiento de las características de ejecución reales*. Debemos asegurarnos que están representadas las características primordiales que pueden surgir en un uso real.
3. *Reducir el espacio de parámetros*. Incluso en el espacio de valores reales de los parámetros, debemos intentar eliminar puntos cuando esto sea posible en función del conocimiento de la aplicación, con el fin de ahorrar tiempo y recursos sin perder mucha cobertura, y para determinar cuándo es necesario un análisis de sensibilidad explícito.

Podemos reducir el espacio en función de los objetivos del estudio, las restricciones en los parámetros impuestos por la tecnología y el entendimiento de las interacciones entre los parámetros.

Comencemos con el proceso de elección de los parámetros usando en el núcleo de resolución de ecuaciones como ejemplo. Aunque examinaremos los parámetros uno por uno, las cuestiones que surjan en etapas posteriores pueden hacernos revisar las decisiones tomadas anteriormente. Empezaremos por elegir el tamaño del problema y el número de procesadores ya que éstos son los que están más limitados por los recursos disponibles.

Tamaño del problema y número de procesadores

Elegimos el tamaño de los problemas y el número de procesadores basándonos en las consideraciones de las características inherentes del programa que hemos discutido al evaluar una máquina real y reduciéndolos para la simulación. Por ejemplo, si el problema es lo bastante grande para que la razón comunicación/cálculo sea muy pequeña, la transferencia de bloques no afectará mucho al rendimiento total; tampoco ayudará si el tamaño del problema es tan pequeño para que el desequilibrio de la carga sea el cuello de botella dominante.

Fijemos ahora el tamaño del problema para la resolución de ecuaciones a una malla de 514×514, el número de procesadores a 16, y examinemos como se eligen el resto de parámetros.

Tamaño de la caché

Como es normal, elegimos el tamaño de las cachés en función del conocimiento de la curva del conjunto de trabajo. Dada un curva del conjunto de trabajo y el conocimiento de como varían los conjuntos de trabajo, el proceso de elegir el tamaño de las cachés para un tamaño de problema dado es análogo al de elegir un tamaño de problema.

Ejemplo La figura 5.11 muestra los conjuntos de trabajo del núcleo de resolución de ecuaciones. ¿Cuantos tamaños de cachés elegirías en este caso?.

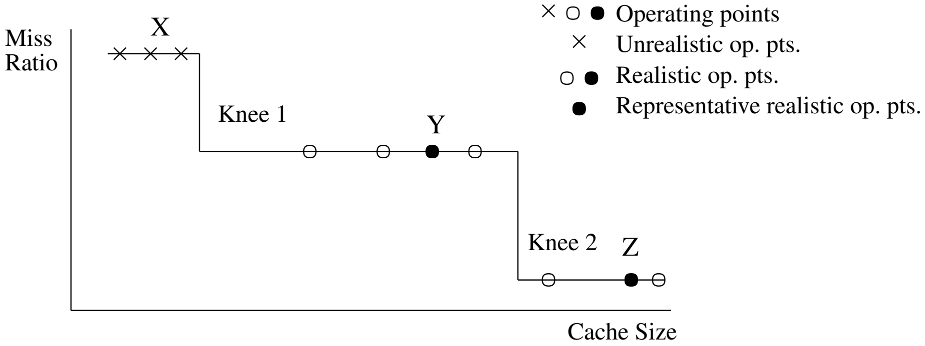
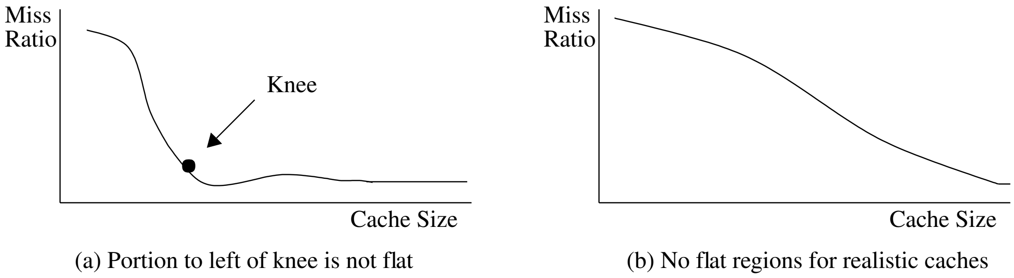


Figura 5.11: Elección del tamaño de la caché para una evaluación usando el núcleo de resolución de ecuaciones.

Respuesta Aunque los tamaños de los conjuntos de trabajo importantes dependen a menudo de los parámetros de la aplicación y el número de procesadores, su naturaleza y, por tanto, la forma general de la curva del conjunto de trabajo no cambia con estos parámetros. Como conocemos el tamaño de cada conjunto de trabajo importante en el núcleo de resolución de ecuaciones y cómo varían con estos parámetros, si conocemos el rango del tamaño de las cachés que son realistas para las máquinas objeto, entonces podemos decir si es (1) irreal esperar que el conjunto de trabajo se ajuste a la caché en situaciones prácticas, (2) irreal esperar que el conjunto de trabajo no se ajuste a la caché, o (3) realista esperar que se ajuste en algunas combinaciones reales de los valores de los parámetros y que no se ajuste para otros[3]. Así, podemos decir cuáles de las regiones entre los codos de la curva pueden ser representativas de situaciones reales y cuales no. Para un tamaño de problema y número de procesadores dado, podemos usar la curva de conjuntos de trabajo para elegir el tamaño de las cachés y evitar regiones no representativas, cubrir las representativas, y recortar en las regiones planas eligiendo un único tamaño de caché para ellas (si todo lo que nos interesa acerca de la caché es su tasa de fallo).

El que un conjunto de trabajo importante se ajuste a la caché afecta sustancialmente a los beneficios de la transferencia por bloques de formas interesantes. El efecto depende de si el conjunto de trabajo consiste en datos almacenados localmente o no. Si consiste principalmente en datos locales —como en el programa encargado de resolver ecuaciones cuando los datos se colocan adecuadamente— pero éstos no caben en la caché, el procesador ocupa la mayor parte de su tiempo parado en el sistema de memoria local. Como resultado, el tiempo de comunicación pasa a ser relativamente poco importante, y la transferencia de bloques es probable que ayude menos (la transferencia de datos mediante bloque también interfiere más con el tráfico local del nodo, causando contención). Sin embargo, si los conjuntos de trabajo son mayoritariamente datos no locales, tenemos el efecto opuesto: si no caben en la caché, entonces existe más comunicación y la probabilidad de que la transferencia de bloques ayude al rendimiento es mayor.



(a) Portion to left of knee is not flat

(b) No flat regions for realistic caches

Figura 5.12: Curvas de la tasa de fallos en función del tamaño de la caché que no consisten en codos abruptos separados por regiones planas.

Por supuesto, la curva del conjunto de trabajo no está siempre compuesta por regiones planas adyacentes separadas por codos bien definidos. Si la curva tiene codos pero las regiones que separan no son planas (ver figura 5.12a), todavía podemos eliminar regiones enteras si sabemos que presentas situaciones irreales. Sin embargo, si una región es realista pero no plana, o si no existe ningún codo hasta que todo el conjunto de trabajo cabe en la caché (como en la figura 5.12b), entonces debemos

---

[3] Si un conjunto de trabajo para un tamaño determinado se ajusta a la caché puede depender de la asociatividad de la caché e incluso quizás del tamaño de bloque, además del tamaño de la caché, pero este hecho no es en la práctica un factor importante si suponemos que la caché es, al menos, asociativa de dos vías. Por tanto, podemos ignorar por ahora estos efectos.

recurrir al análisis de sensibilidad, eligiendo puntos próximos a los extremos al igual que, quizás, algunos intermedios. De nuevo, una evaluación adecuada requiere que entendamos bien las características claves de las aplicaciones.

La pregunta pendiente es cómo determinar la posición de los codos y la forma de la curva de trabajo entre ellos. En casos simples, podemos hacerlo de forma analítica. Sin embargo, los algoritmos son complejos, los factores constantes son difíciles de predecir y los efectos del tamaño de bloque de la caché y la asociatividad pueden ser difíciles de analizar. En estos casos, obtenemos la curva para un tamaño de problema dado y número de procesadores mediante simulación con diferentes tamaños de caché. Las simulaciones necesarias son relativamente poco costosas ya que el tamaño de los conjuntos de trabajo no depende detalladamente de factores relacionados con el tiempo como las latencias, ocupaciones y contención, que no necesitan simularse de forma cuidadosa. A continuación se puede analizar o medir cómo los conjuntos de trabajo cambian con el tamaño del problema o el número de procesadores. Afortunadamente, el análisis de ratios de crecimiento es, normalmente, más fácil de predecir que los factores constantes. Además, aspectos de bajo nivel como el tamaño de bloque y asociatividad no cambian, a menudo, los conjuntos de trabajo para cachés suficientementes grandes y organizaciones de cachés razonables (otras distintas a las cachés de mapeo directo).

Tamaño de bloque de caché y asociatividad

Además del tamaño del problema, el número de procesadores y el tamaño de la caché, el tamaño del bloque de la caché es otro parámetro importante para determinar los beneficios de la transferencia de bloques. Los resultados, sin embargo, son un poco más complicados. Los bloques de cachés grandes actúan por sí mismos como transferencias de bloques pequeños para programas con buena localidad espacial, haciendo la transferencia explícita de bloques menos efectiva en estos casos. Por otra parte, si la localidad espacial es pobre, entonces el tráfico extra causado por bloques de cachés grandes (debido a la fragmentación o falsa compartición) puede consumir más ancho de banda que el necesario cuando la comunicación se realiza a través de lecturas y escrituras. El que la pobre localidad espacial malgaste ancho de banda para la transferencia segmentada de bloques depende también de si la transferencia se implementa enviando los bloques completos de la caché a través de la red o únicamente las palabras necesarias. Obsérvese que la propia transferencia de bloques incrementa las necesidades de ancho de banda ya que hace que la misma cantidad de comunicación que se realice en menos tiempo. Así, si la transferencia de bloque se implementa usando transferencias segmentadas de bloques de cachés y la localidad espacial es pobre, usar la transferencia de bloque puede dañar, en lugar de ayudar, cuando el ancho de banda disponible es limitado ya que puede incrementar la contención para el ancho de banda disponible.

Afortunadamente, normalmente podremos restringir el conjunto de tamaños de caché de interés ya sea debido a restricciones de la tecnología actual o debido a limitaciones impuestas por el conjunto de bloques de construcción disponibles. Por ejemplo, casi todos los microprocesadores actuales soportan bloques de cachés entre 32 y 128 bytes, y podemos haber elegido ya un procesador que tenga un bloque de caché de 64 bytes. Cuando existen umbrales en la interacción del tamaño del problema y el tamaño del bloque de la caché (por ejemplo, en el caso de ordenación radix que veremos posteriormente), debemos asegurarnos que cubrimos ambos lados del umbral.

Mientras que la magnitud del impacto de la asociatividad de las cachés es muy difícil de predecir, las cachés reales se construyen con una asociatividad baja (a lo más, asociativas de 4 vías), con lo que el conjunto de posibilidades a considerar es pequeño. Si elegimos una única asociatividad, debemos evitar cachés de mapeo directo (al menos al más bajo nivel de la jerarquía, la más lejana del procesador) a no ser que sepamos que las máquinas de interés las tienen.

Parámetros de rendimiento de la arquitectura de las comunicaciones

Una vez discutidos los parámetros de organización de la jerarquía de memoria extendida, consideremos los parámetros claves de rendimiento de la arquitectura de las comunicaciones —sobrecarga, retraso en la red o tiempo de tránsito y ancho de banda— y cómo afectan a los beneficios de la transferencia de bloques. Debemos elegir los valores base para estos parámetros en función de lo que esperemos para el sistema real, pero el entendimiento del impacto de los mismos nos ayuda a entender qué parámetros deben variarse y cómo.

A mayor peso del componente de la sobrecarga en una comunicación de un bloque de caché (por ejemplo, en un fallo de caché), más importante es amortizar esta sobrecarga estructurando la comunicación en transferencias de bloques mayores. Esto es cierto siempre que la sobrecarga de iniciar la transferencia de un bloque no sea tan alta como para eliminar los beneficios, dado que la sobrecarga de una transferencia de bloque iniciada de forma explícita puede ser mayor que la transferencia de un bloque de caché iniciada implícitamente.

Por la misma causa, a mayor latencia entre nodos de la red, mayor será el beneficio de amortizarla mediante transferencias de bloques grandes. Los efectos de cambiar la latencias no muestran, normalmente, codos o umbrales, por lo que para examinar un intervalo de posibles latencias, debemos realizar un análisis de sensibilidad eligiendo algunos puntos de dicho intervalo. En la práctica, elegiremos latencias basándonos en las latencias de las máquinas de interés; por ejemplo, los multiprocesadores fuertemente acoplados tienen latencias mucho más pequeñas que las estaciones de trabajo en una red de área local.

El ancho de banda disponible también es una cuestión importante en el estudio de la transferencia de bloques. El ancho de banda también muestra un fuerte efecto *codo*, que es de hecho un efecto de saturación: si existe suficiente ancho de banda disponible para las necesidades de la aplicación, o si no es así. Si ocurre lo primero, no importa si el ancho de banda disponible es cuatro veces el necesario o diez veces. Por tanto, podemos elegir un ancho de banda que sea menor que el necesario y uno que se mucho mayor. Dado que el estudio de la transferencia por bloques es particularmente sensible al ancho de banda, podemos elegir también un valor cercano a la frontera. En la elección de los valores del ancho de banda, debemos considerar cuidadosamente las ráfagas en las demandas de ancho de banda por parte de la aplicación; el ancho de banda medio necesario durante toda la aplicación puede ser pequeño, pero la aplicación puede saturar un ancho de banda mayor durante periodos de comunicaciones intensivas, dando lugar a contención.

Revisión de las elecciones

Para finalizar, a menudo deberemos revisar nuestras primeras elecciones para el valor de los parámetros en función de las interacciones con parámetros considerados posteriormente. Por ejemplo, si estamos obligados a utilizar tamaños de problema pequeños debido a la falta de tiempo de simulación o recursos, entonces podemos estar tentados a elegir tamaños de cachés muy pequeñas para representar situaciones reales cuando un conjunto de trabajo importante no cabe en la caché. Sin embargo, elegir una caché muy pequeña puede dar lugar a artificios severos, especialmente si usamos cachés de mapeo directo o un tamaño de bloque de caché grande (ya que esta situación dará lugar a muy pocos bloques en la caché y, potencialmente, a mucha fragmentación y conflictos de mapeo). Por tanto, debemos reconsiderar nuestra elección del tamaño del problema y número de procesadores para la cual queremos representar esta situación.

Resumiendo, la argumentación que hemos realizado a lo largo de esta sección nos muestra que los resultados de un estudio de evaluación pueden ser erróneos si no cubrimos el espacio adecuadamente: podemos elegir una combinación de parámetros y cargas de trabajo que demuestren buenos beneficios en el rendimiento para una característica como la transferencia de bloques (por ejemplo, un tamaño de problema relativamente pequeño, grandes cachés y un tamaño de bloque de caché pequeño), y podemos elegir igual de fácilmente una combinación que no lo demuestre. Por tanto, es muy importante que incorporemos guías metodológicas sólidas en nuestros estudios de la arquitectura y entender las interacciones relevantes entre el hardware y el software.

5.4 Ilustración de la caracterización de cargas de trabajo

Las cargas de trabajo se usan intensivamente para ilustrar cuantitativamente algunas de las decisiones arquitecturales y para evaluar máquinas reales. En particular, para ilustrar este punto nos centraremos en los programas escritos para un espacio de memoria compartida con coherencia de cachés y concretamente en seis aplicaciones paralelas y núcleos computacionales que se ejecutan en modo de trabajo por lotes (e.d., uno a la vez) y que no incluyen la actividad del sistema operativo, y un carga de trabajo multiprogramada que incluye dicha actividad. Mientras que el número de cargas de trabajo que utilizaremos es pequeña, las aplicaciones representan importantes clases de cálculos y presentan características que varían ampliamente.

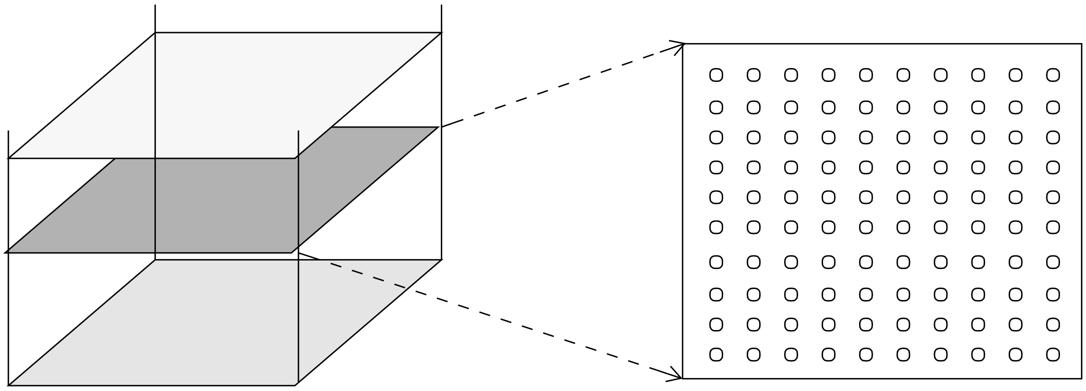
5.4.1 Estudio de casos de cargas de trabajo

Todos los programas paralelos que usaremos para arquitecturas de espacio de direcciones compartido están incluidas en el conjunto de aplicaciones SPLASH-2. En esta sección describiremos brevemente algunas de las cargas de trabajo: Ocean, Barnes-Hut, Raytrace, LU, Radix, Radiosity y Multiprog. Ocean, Barnes-Hut, Raytrace y Radiosity son aplicaciones reales, LU y Radix son núcleos computacionales, y Multiprog es una carga de trabajo multiprogramada. En la sección 5.4.2, mediremos algunas características de ejecución metodológicamente relevantes de las cargas de trabajo, incluyendo una descomposición del acceso a los datos, el ratio comunicación/cálculo y cómo se escala, y el tamaño y el escalamiento de los conjuntos de trabajo más importantes. Usaremos esta caracterización para elegir los parámetros del sistema de memoria para estas aplicaciones y el conjunto de datos en posteriores capítulos.

Ocean

Para modelizar el clima de la Tierra es importante entender cómo la atmósfera interactúa con los océanos, que ocupan las 3/4 partes de la superficie de la Tierra. La aplicación Ocean simula las corrientes de agua en el océano. Estas corrientes se desarrollan y evolucionan bajo varias fuerzas físicas, incluyendo los efectos atmosféricos, el viento y la fricción con la base del océano. El objetivo de esta aplicación es simular la evolución de estas corrientes a lo largo del tiempo y entender sus interacciones con el flujo medio del océano.

El problema físico real es continuo en dos espacios (la cuenca del océano y el tiempo), pero para permitir la simulación mediante ordenador discretizaremos el problema en ambas dimensiones. Para discretizar el espacio, modelamos la cuenca del océano como una malla de puntos. Cada variable importante — como la presión, velocidad y varias corrientes— tiene un valor en cada punto de la malla. Esta aplicación no utiliza una representación tridimensional sino un conjunto de representaciones en dos dimensiones de secciones horizontales a través de la cuenca del océano, cada una de las cuales está representada por una malla bidimensional de $(n + 2) \times (n + 2)$ puntos[4] (ver figura 5.13). Para la dimensión temporal, la aplicación discretiza el tiempo en un serie finita de pasos de tiempo. Las ecuaciones de movimiento se resuelven para todos los puntos de la malla en un paso concreto, el estado de las variables se actualizan como resultado de ese cálculo y de nuevo se pasan a resolver las ecuaciones de movimiento para el siguiente paso de tiempo.



(a) Cross-sections

(b) Spatial discretization of a cross-section

Figura 5.13: Secciones transversales del océano y su discretización mediante mallas regulares.

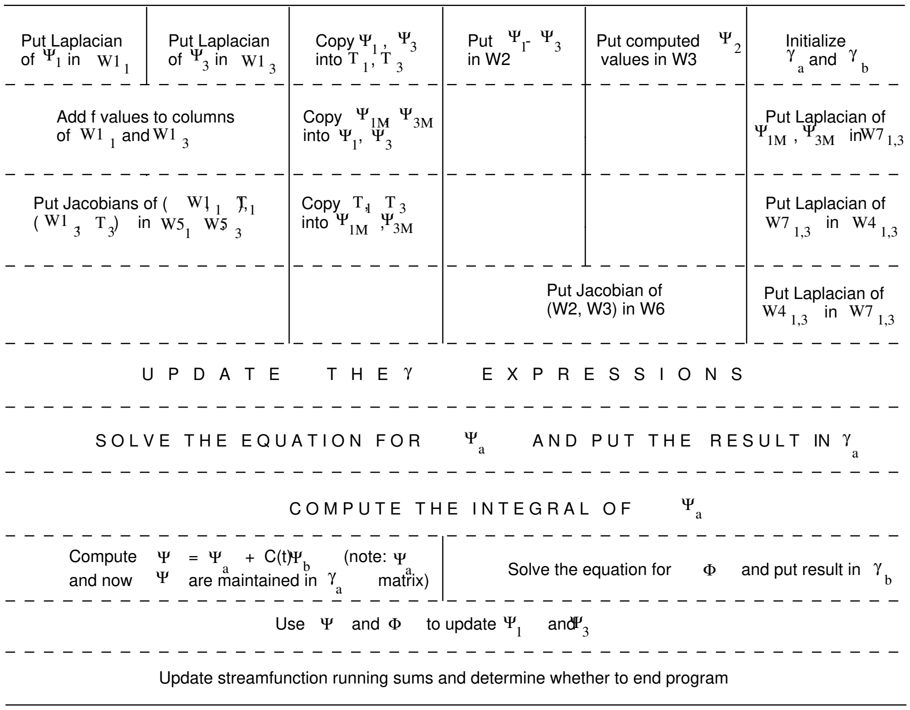
Cada paso de tiempo consta de 33 cálculos de malla diferentes, cada uno de los cuales afecta a una o varias mallas (variables). Los cálculos típicos incluyen sumar los resultados de unas pocas mallas multiplicados por un valor escalar y almacenarlo en otra malla (p.e., $A = \alpha_1 B + \alpha_2 C - \alpha_3 D$), realizar para cada elemento la media con sus vecinos más cercanos y almacenar el resultado en otra malla y resolver un sistema de ecuaciones diferenciales parciales sobre una malla utilizando un método iterativo.

Ocean afronta la concurrencia dentro de un paso de tiempo a dos niveles: a lo largo de los cálculos que ser realizan en una malla (paralelismo funcional) y dentro de un cálculo concreto sobre la malla (paralelismo de datos). La figura 5.14 muestra la estructura de las dependencias entre las distintas operaciones que se realizan en un paso de tiempo. Claramente, no existe suficiente concurrencia entre los diferentes cálculos (e.d, no existen suficientes secciones verticales) para ocupar más que unos pocos procesadores. Por tanto, debemos concentrarnos en explotar el paralelismo de datos existente y decidir qué combinación de paralelismo de datos y funcional es el mejor.

En este caso, elegimos que todos los procesos colaboren en cada cálculo que afecte a una malla en

---

[4] Se usan $n + 2$ puntos en lugar de n para que el número de puntos internos que se actualizan mediante la resolución de ecuaciones sea $n \times n$.



Note: Horizontal lines represent synchronization points among all processes, and vertical lines spanning phases demarcate threads of dependence.

Figura 5.14: Ocean: Fases en un paso de tiempo y dependencias entre los cálculos.

lugar de dividirlos entre todos los cálculos disponibles en la malla actual y usar ambos niveles de paralelismo. La combinación del paralelismo funcional y de datos incrementaría el tamaño de la partición de cada proceso en la malla, reduciendo la razón comunicación/cálculo. Sin embargo, el trabajo asociado con los diferentes cálculos que se realizan en la malla son muy variados y dependen también del tamaño del problema, lo que complica el equilibrio de la carga. En segundo lugar, dado que en un paso de tiempo varios cálculos deben acceder a la misma malla, no sería conveniente dividir de forma diferente la malla entre los distintos procesos para los diferentes cálculos, ya que ello afectaría a la localidad de las comunicaciones y datos. En tercer lugar, todos los cálculos que se realizan en la red son totalmente paralelizables, y todos los puntos de la malla en un determinado cálculo realizan aproximadamente el mismo valor, por lo que podemos asignar de forma estática puntos de la malla a procesos. No obstante, conocer qué cálculos de la malla son independientes es útil ya que permite a los procesos sincronizarse entre ellos (ver figura 5.14).

Barnes-Hut

La segunda aplicación en SPLASH-2 busca entender la evolución de las estrellas en un sistema de galaxias a lo largo del tiempo. Por ejemplo, podemos querer estudiar qué pasa cuando dos galaxias colisionan o cómo a partir de un conjunto aleatorio de estrellas se llega a una galaxia de forma definida. Este problema implica la simulación del movimiento de un conjunto de cuerpos (en este caso estrellas) moviéndose bajo las fuerzas que ejercen unos sobre otros. A este tipo de problema se le denomina el problema de n-cuerpos. El cálculo se discretiza en el espacio tratando cada estrella como un cuerpo separado o utilizando un cuerpo para representar muchas estrellas. También se discretiza el cálculo del tiempo mediante la simulación de las galaxias en diferentes pasos de tiempo. En cada uno de ellos se calcula la fuerza gravitatoria ejercida sobre cada estrella por el resto y se actualiza su posición.

El cálculo de las fuerzas existentes entre las estrellas es la parte más costosa de cada iteración. Si calculásemos las interacciones entre todas las estrellas, obtendríamos una de complejidad de $O(n^2)$. Dado que este hecho hace que la simulación para sistemas grandes sea imposible, podemos utilizar métodos jerárquicos basados en árbol para reducir la complejidad a $O(n \log n)$ para distribuciones generales, o incluso $O(n)$ para distribuciones uniformes. Los algoritmos jerárquicos se basan en el hecho de que la interacción gravitacional decae con las distancia en función de

$$G\frac{m_1 m_2}{r^2}$$

por lo que si un grupo de estrellas está lo suficientemente lejos de una estrella dada, podemos calcular el efecto del grupo sobre la estrella aproximando el grupo a una única estrella en el centro del grupo con poca pérdida de exactitud.

Para facilitar una aproximación jerárquica, el algoritmo Barnes-Hut representa el espacio tridimensional que contiene las galaxias en un árbol. La raíz del árbol representa un espacio de celdas que contiene todos los cuerpos del sistema. El árbol se construye añadiendo cuerpos a la raíz que inicialmente se encuentra vacía y subdividiendo una celda en ocho hijos del igual tamaño tan pronto como contenga más de un número fijo de cuerpos. El resultado es un árbol octario cuyos nodos internos son las celdas en que se ha dividido el espacio y cuya hojas contienen cuerpos individuales. La figura 5.15 muestra un ejemplo para un dominio bidimensional y el correspondiente árbol cuaternario. Las posiciones de los cuerpos cambian a lo largo de los diferentes instantes de tiempos, por lo que el árbol se reconstruye en cada iteración. La estructura del cómputo global se muestra en la figura 5.16, siendo la fase de cálculo de las fuerzas la que ocupa la mayor parte del tiempo.

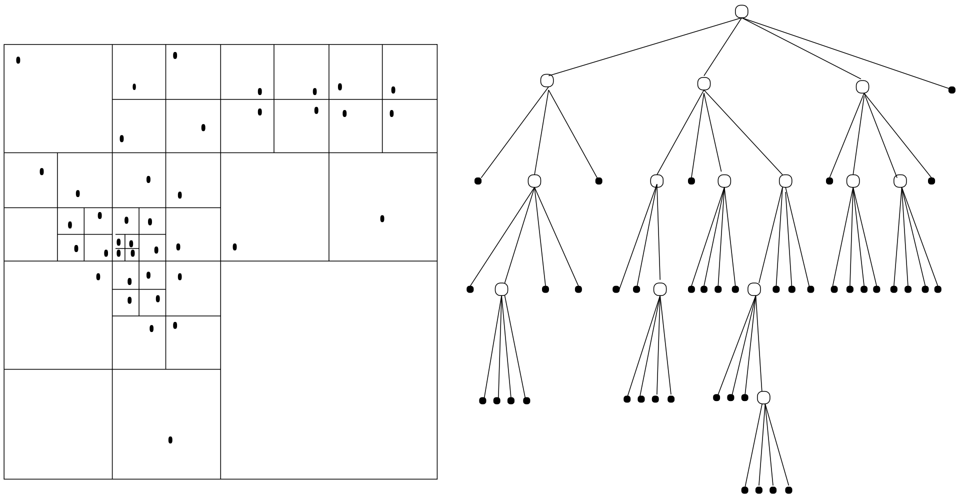


Figura 5.15: Una distribución bidimensional de partículas y su correspondiente árbol cuaternario en Barnes-Hut.

El árbol se atraviesa una vez por cuerpo para calcular el conjunto de fuerzas que actúan sobre dicho cuerpo. El algoritmo de cálculo de fuerzas para un cuerpo comienza en la raíz del árbol y se realiza el siguiente test de forma recursiva para cada celda que visita. Si el centro de gravedad de la celda está lo suficientemente lejano del cuerpo, el subárbol bajo esa celda se aproxima por un único cuerpo en el centro de gravedad de la celda. El resto de ese subárbol no se recorre. Si, por el contrario, el centro de gravedad no está lo suficientemente lejos, debemos visitar cada una de las subceldas. La condición para determinar si una celda está lo suficientemente lejos es:

$$\frac{l}{d} < \theta$$

donde l es la longitud del lado de la celda, d es la distancia del cuerpo al centro de gravedad de la celda y θ es un parámetro de precisión definido por el usuario (θ suele estar entre 0.5 y 1.2). Dado que la profundidad del árbol será $O(logn)$ y el número de cuerpos para el que se debe atravesar el árbol es n, la complejidad del algoritmo es $O(nlogn)$. En realidad es

$$O\left(\frac{1}{\theta^2}nlogn\right)$$

ya que θ determina el número de celdas del árbol que se visitan en cada nivel (un valor menor de θ implica mayor exactitud y más celdas que deben visitarse).

Conceptualmente, la principal estructura de datos en la aplicación es el árbol Barnes-Hut. El árbol se implementa tanto en el programa secuencial como en el paralelo mediante dos arrays: un array de cuerpos y un array de celdas del árbol. Cada cuerpo y celda se representa como una estructura o registro. Los campos para un cuerpo incluyen su posición tridimensional, velocidad, aceleración y centro de gravedad tridimensional. Existe un array separado de punteros a los cuerpos y uno de punteros a las celdas. A cada proceso le corresponde un trozo de esos arrays de punteros, no necesariamente del mismo tamaño, que en cada paso de tiempo se asigna a aquellos cuerpos y celdas asignados al proceso en ese instante de tiempo. Dado que la estructura y el particionamiento del árbol cambia a lo largo del tiempo conforme evolucionan la galaxia, las celdas y cuerpos reales asignados a un proceso no están contiguos en los arrays de cuerpos y celdas.

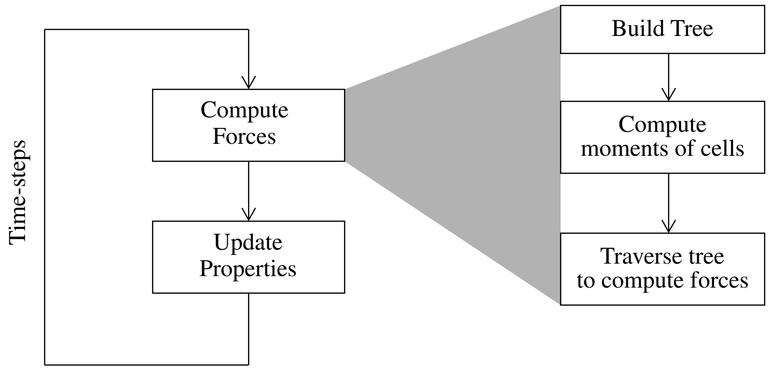


Figura 5.16: Flujo de cálculo en la aplicación Barnes-Hut

Raytrace

La tercera aplicación procede de la visualización de escenas complejas en el campo de gráficos por ordenador. Una técnica habitual para representar estas escenas en imágenes consiste en la técnica de *trazado de rayos*.

La escena se representa mediante un conjunto de objetos en un espacio tridimensional, representándose la imagen como un array bidimensional de pixels cuyos valores de color, opacidad y brillo deben calcularse. La escena se representa como vista desde un determinado punto de vista o posición del ojo. El programa envía rayos desde ese punto de vista a través de cada pixel del plano que forma la imagen hasta la escena —calculando su reflexión, refracción e interacciones luminosas conforme golpean y se reflejan en los objetos— para posteriormente calcular los valores de color y brillo de los correspondientes pixels.

El algoritmo usa un representación jerárquica del espacio denominado Malla Uniforme Jerárquica (Hierarchical Uniform Grid o HUG), que tiene una estructura similar al árbol octario usado en la aplicación Barnes-Hut. La raíz del árbol representa el espacio total que engloba la escena, cada hoja contiene una lista enlazada de primitivas de objeto (el máximo número de primitivas por hoja lo define el usuario, al igual que otros aspectos de la estructura del árbol). Esta malla jerárquica permite saltarse de forma eficiente regiones vacías cuando se traza un rayo.

LU

La factorización LU es el proceso de convertir una matriz densa A en dos matrices L, U que son triangulares inferior y superior, respectivamente, y cuyo producto es igual a A (e.d., $A = LU$). Su utilidad está en la resolución de sistemas de ecuaciones lineales y se puede encontrar en aplicaciones científicas al igual que en métodos de optimización como la programación lineal.

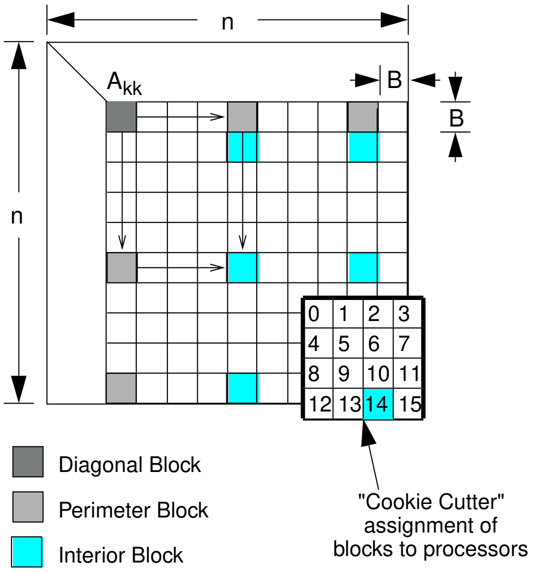
La factorización LU funciona como la eliminación por el método de Gauss. La complejidad computacional de la factorización LU es $O(n^3)$, mientras que el tamaño de los datos es $O(n^2)$. Esta situación es ideal para explotar la localidad temporal mediante la utilización de bloques. De hecho, utilizaremos una factorización LU por bloques, que es mucho más eficiente la versión sin bloques tanto en su forma secuencial como paralela. Para factorizar la matriz $n \times n$ se divide en bloques de $B \times B$, y la idea es reutilizar un bloque tanto como sea posible antes de pasar al siguiente bloque. A partir de ahora pensaremos en una matriz consistente en $\frac{n}{B} \times \frac{n}{B}$ bloques en lugar de en $n \times n$ elementos y elimina y actualiza los bloques uno cada vez como si fueran elementos. Las operaciones matriciales como la multiplicación y la inversión se usan sobre los bloques de $B \times B$ en lugar de operaciones escalares sobre elementos. El seudocódigo secuencial para esta descomposición se muestra en la figura 5.17.

```
for k ← 0 to N-1 do                              /*loop over all diagonal blocks */
    factorize block $A_{k,k}$;
    for j ← k+1 to N-1 do                        /* for all blocks in the row of, and
                                                  to the right of, this diagonal block */
        $A_{k,j} \leftarrow A_{k,j} * (A_{k,k})^{-1}$;          /* divide by diagonal block */
        for i ← k+1 to N-1 do                    /* for all rows below this diagonal block*/
            for j ← k+1 to N-1 do                /* for all blocks in the corr. row */
                $A_{i,j} \leftarrow A_{i,j} - A_{i,k} * (A_{k,j})^T$;
            endfor
        endfor
    endfor
endfor
```

Figura 5.17: Seudocódigo describiendo la factorización secuencial densa LU por bloques.

Consideremos los beneficios de la utilización de bloques. Si no utilizaramos bloques en los cálculos, un procesador calcularía un elemento, a continuación calcula el siguiente elemento a su derecha, y continua así hasta que termina la fila actual, después de lo cual procede con la siguiente fila. Cuando vuelve al primer elemento activo de la siguiente fila, estaría referenciando un elemento del perímetro de la fila (el utilizado para calcular el correspondiente elemento activo de la fila previa). Sin embargo, en este momento ha estadp manejando datos proporcionalmente a una fila entera de una matriz y, para matrices grandes, el elemento del perímetro de la fila puede que ya no esté en la caché. En la versión con bloques, en cada iteración del bucle más interior de los cálculos (es decir, el cálculo de la línea $A_{i,j} \leftarrow A_{i,j} - A_{i,k} * (A_{k,j})^T$), actúa únicamente sobre *B* elementos en una dirección antes de volver a los datos previamente referenciados que todavía están en la caché y pueden reutilizarse. Las operaciones (multiplicaciones de matrices y factorizaciones) sobre bloques de $B \times B$ tienen una complejidad de $O(B^3)$ en cálculos y accesos a los datos, en donde cada elemento del bloque se accede B veces. Si el tamaño del bloque se elige de tal manera que un bloque de $B \times B$ o B^2 elementos (más algunos datos adicionales) cabe en la caché, entonces en un cálculo de bloques dado solamente el primer acceso a un elemento produce un fallo en la caché. Los accesos posteriores producen acierto en la caché, dando lugar a B^2 fallos para B^3 accesos o una tasa de fallo de $\frac{1}{B}$.

En la versión paralela, podemos pensar como tarea en cada cálculo que actualiza un bloque. La figura 5.18 proporciona una representación pictórica del flujo de información entre los bloques dentro de la iteración del bucle exterior y muestra cómo asignamos los bloques (y, por tanto, las tareas) a los procesadores en la versión paralela. Debido a la naturaleza de los cálculos, los bloques de la parte superior izquierda de la matriz están activos únicamente en las primeras iteraciones del bucle exterior, mientras que los bloques cercanos a la parte inferior derecha tienen más trabajo asociado. Asignar filas contiguas o cuadrados de bloques a procesadores daría lugar a un pobre equilibrio de la carga. Por lo tanto, intercalamos bloques entre los procesadores en ambas dimensiones, dando lugar a un particionamiento denominado *descomposición dispersa* de la matriz: los procesadores se ven como formando una malla bidimensional de $\sqrt{p} \times \sqrt{p}$, y esta malla de procesadores se estampa de forma



```
for all k from 0 to N-1
   if I own $A_{kk}$, factorize $A_{kk}$
   BARRIER;
   for all my blocks $A_{kj}$ in pivot row/col
         $A_{kj}$ <- $A_{kj}$*$A_{kk}^{-1}$
   BARRIER;
   for all my blocks Aij in active interior
         $A_{ij}$ <- $A_{ij}$ - $A_{ik}$*$A_{kj}$
endfor
```

Figura 5.18: Factorización paralela de LU por bloques: flujo de información, particionamiento y pseudocódigo paralelo.

repetitiva sobre los bloques de cálculo como en una cortadora de galletas.

La desventaja de dividir los cálculos en bloques en lugar de elementos individuales es que incrementa la granularidad de la tarea y daña el equilibrio de carga: se reduce la concurrencia ya que hay menos bloques que elementos. En la factorización secuencial de LU, el único límite al tamaño del bloque es que los dos o tres bloques utilizados en el cálculo de un bloque puedan estar en la caché. En el caso paralelo, el tamaño de bloque ideal viene determinado por el compromiso entre la localidad de los datos y la sobrecarga de las comunicaciones (particularmente, en el caso de máquinas de paso por mensaje) que hacen que los bloques sean mayores y el equilibrio de carga que hace que los bloques sean menores. Por tanto, el tamaño de bloque ideal depende del tamaño del problema, número de procesadores y otros parámetros arquitecturales. En la práctica, para máquinas paralelas grandes funcionan bien tamaños de bloque de 16×16 ó 32×32 elementos.

Radix

El programa radix ordena un conjunto de enteros, denominados *claves*, usando el método de ordenación *base*. Supogamos que tenemos *n* enteros a ordenar, cada uno de un tamaño de *b* bits. El algoritmo utiliza una base de r bits, donde r la elige el usuario. Estos significa que los b bits que representan una clave pueden verse como un conjunto de $\lceil b/r \rceil$ grupos de r bits cada uno (ver la figura 5.19). El algoritmo procede en $\lceil b/r \rceil$ fases o iteraciones. Cada fase, comenzado por el grupo de menor orden, ordena las claves en función de sus valores en el correspondiente grupo de r bits, denominado *dígito*. Las claves estarán completamente ordenadas después de estas $\lceil b/r \rceil$ fases. En cada fase se utilizan dos arrays unidimensionales de n enteros: uno, denominado *array de entrada*, almacena las llaves tal como aparecen al inicio de cada fase, y el otro, denominado *array de salida*, almacena las claves una vez finalizada la fase.

Consideremos el cálculo paralelo dentro de una fase. El algoritmo paralelo divide las n claves entre los

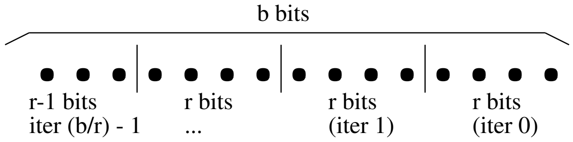


Figura 5.19: Un número de b bits dividido en $\lceil b/r \rceil$ grupos de r bits.

p procesadores de tal manera que al primer procesador se le asigna las primeras $\frac{n}{p}$ claves, al proceso 2 las siguientes $\frac{n}{p}$ y, así, sucesivamente. En cada fase, cada procesador realiza los siguientes pasos:

1. Se realiza una pasada sobre las $\frac{n}{p}$ claves locales para construir un histograma local de los valores de la clave. El histograma tiene 2^r entradas, donde r es el número de bits en un dígito. Si una clave tiene el valor i en la fase actual, se incrementa el i-ésimo bit del histograma.

2. Cuando todos los procesadores han finalizado el paso 1 (determinado por una sincronización a través de una barrera), se acumulan los histogramas locales en un histograma global. El histograma global mantiene la información de cuántas claves hay para cada valor del dígito actual y además, para cada uno de los p valores de j (identificador de proceso), cuántas claves de un valor dado pertenecen a los procesos cuyos identificadores son menores que j.

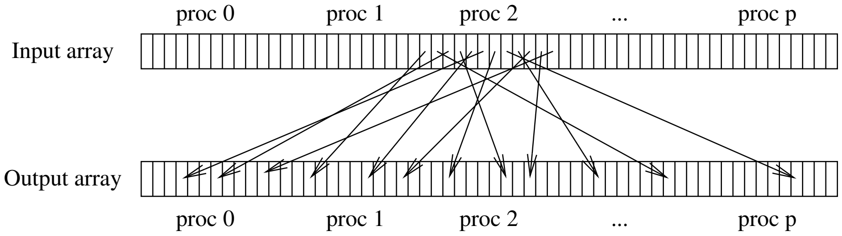


Figura 5.20: El paso de permutación de una fase del ordenamiento base.

3. Realizar otra pasada por las $\frac{n}{p}$ claves locales. Para cada clave, usar el histograma global y los histogramas locales para determinar a qué posición del array de salida debe ir la clave, y escribir el valor de la clave en esa entrada del array de salida. Obsérvese que el elemento del array en donde deberá escribirse el resultado puede ser muy probablemente no local, con una probabilidad esperada de $\frac{p-1}{p}$ (ver figura 5.20). A este paso se le denomina paso de *permutación*.

Radiosity

El método "radiosidad" se utiliza en el área de gráficos por ordenador para calcular la iluminación global en una escena que contiene superficies reflectantes difusas. En el método "radiosidad" jerárquico, una escena se modela inicialmente como k polígonos de entrada, o *retazos*. Por ejemplo, la parte superior de una mesa o la posterior de una silla puede ser un retazo de la entrada. A continuación se calculan las interacciones de la luz entre cada par de ellos. En una visión simplificada del algoritmo, si la luz transferida entre un par de elementos es mayor que un cierto umbral, uno de ellos (digamos, el más grande) se subdivide, y las interacciones se calculan recursivamente entre los retazos resultantes y el otro retazo. Este proceso continúa hasta que la luz transferida entre todos los pares es

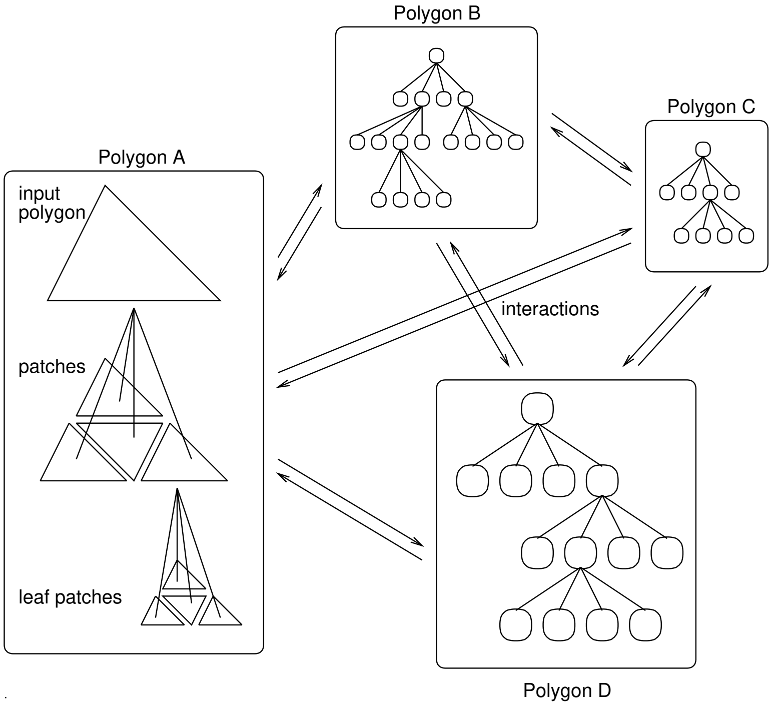


Figura 5.21: Subdivisión jerárquica de los polígonos de entrada en árboles cuaternarios en radiosity.

suficientemente baja. Cada subdivisión da lugar a cuatro nuevos trozos. Si el resultado final de trozos sin dividir es n, entonces con k retazos originales la complejidad del algoritmo es $O(n + k^2)$.

Los retazos de la entrada que conforman la escena se insertan en primer lugar en un árbol de particionamiento del espacio binario (BSP tree), que es una estructura de datos que facilita el cálculo eficiente de la visibilidad entre cada par de trozos. Cada elemento inicial tiene una *lista de interacción* de los otros elementos que son potencialmente visibles desde él y con los cuales debe calcular las interacciones. A continuación, se calculan las "radiosidades" mediante el siguiente algoritmo iterativo:

1. Para cada retazo de la entrada, calcular su "radiosidad" debida a todos los elementos que están en su lista de interacción, subdividiéndose a si mismo o a otros retazos y calculando sus interacciones de forma recursiva si es necesario (ver figura 5.21).
2. Comenzando por los retazos de las hojas del árbol, sumar todos los valores calculados (ponderados por sus áreas) para obtener el valor final de "radiosidad" de la escena, y compararlo con la iteración previa en busca de convergencia dentro de una tolerancia fija. Si la "radiosidad" ha convergido, volver al paso 1.
3. Procesar la solución para mostrarla.

La mayor parte del tiempo de una iteración se consume en el paso 1, por lo que vamos a analizarlo más en profundidad. Supongamos que un retazo i esta atravesando su lista de interacciones para calcular la interacciones con otros retazos. La interacción con otro trozo, digamos j, incluye calcular la intervisibilidad de dos trozos a la vez que la luz que se transfiere entre ellos. (La transferencia de luz real es el producto de la intervisibilidad real y de la transferencia de luz que pudiera ocurrir si no hubiese oclusión y hubiese total intervisibilidad). Calcular la intervisibilidad implica atravesar el árbol BSP varias veces de un retazo a otro; de hecho, el cálculo de la visibilidad es una porción muy grande del tiempo de ejecución total. Si el resultado de una interacción dice que el retazo "origen", i, debe subdividirse, entonces se crean cuatro elementos hijos, si estos ya no existían debido a una interacción previa; el retazo j se elimina de la lista de interacción de i y se añade a cada una de las listas de interacciones de los hijos de i. Si el resultado es que se debe dividir el retazo j, entonces se sustituye j por sus hijos en la lista de interacciones de i. Esto significa que las interacción a calcular a continuación son las que existen entre i y cada uno de los hijos de j. Estas nuevas interacciones pueden ser, a su vez, causa de nuevas subdivisiones, de forma que el proceso continua recursivamente. La figura 5.22 muestra un ejemplo de este refinamiento jerárquico iterativo.

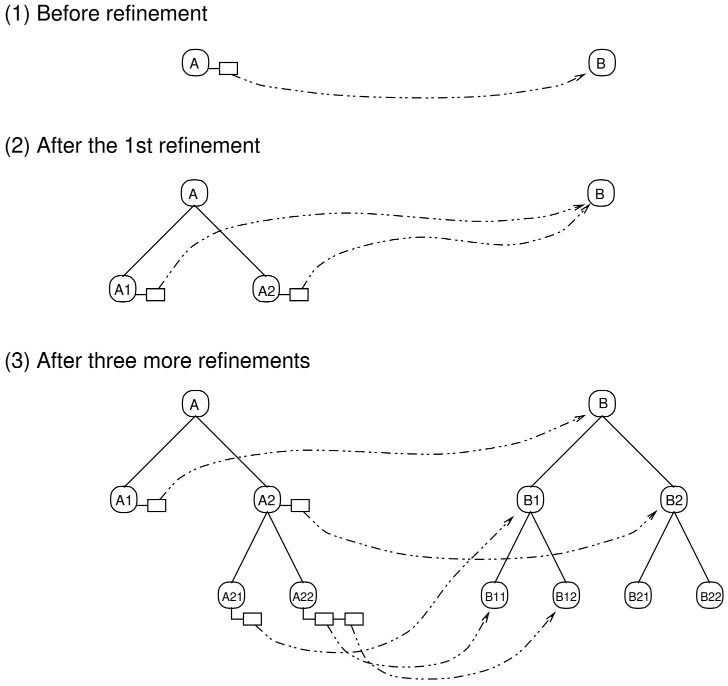


Figura 5.22: Refinamiento jerárquico de las interacciones y listas de interacción.

El paralelismo está disponible a tres niveles en esta aplicación: en los k polígonos entradas, en los retazos en que estos polígonos se subdividen y en las interacciones calculadas para un retazo. Los tres niveles susponen comunicación y sincronización entre procesadores. Obtenemos el mejor rendimiento definiendo una tarea que sea o bien un retazo y todas sus interacciones, o bien una única interacción entre dos retazos, dependiendo del tamaño del problema, el número de procesadores y las características de la máquina.

Dado que los cálculos y las subdivisiones son altamente impredecibles, usaremos colas de tareas y robo de tareas para obtener una carga de trabajo equilibrada. La implementación paralela proporciona a cada procesador su propia cola de tareas. Inicialmente, la cola de un procesador contiene un subconjunto de las interacciones disponibles entre los polígonos. Cuando un retazo se subdivide debido a una interacción, se encolan en la cola de tareas del procesador que calcula la interacción las nuevas tareas para los subretazos. Un procesador ejecuta las tareas de su cola hasta que no quedan tareas. Entonces roba tareas de las colas de otros procesadores. El algoritmo paralelo es no determinista debido al robo de tareas y al orden en que se calculan las interacciones y subdivisiones, presentando patrones de comunicaciones y acceso a los datos impredecibles.

Multiprog

Las cargas de trabajo que hemos discutido incluyen, hasta ahora, únicamente aplicaciones paralelas que se ejecutan en solitario. Sin embargo, el uso habitual de los multiprocesadores, sobre todo en los multiprocesadores de espacio compartido de pequeña escala, es el de mejorar el rendimiento para cargas de trabajo multiprogramado. La compartición de recursos de grano fino que permite estas máquinas hace posible que un único sistema operativo dé servicio a varios procesadores de forma eficiente. La actividad del sistema operativo es, a menudo, un componente sustancial de esas cargas de trabajo, siendo por sí mismo una aplicación paralela y compleja. La última carga de trabajo que estudiaremos es una carga de trabajo multiprogramada, consistente en un conjunto de aplicaciones secuenciales y el propio sistema operativo. Las aplicaciones son dos tareas con el programa *compress* y dos compilaciones paralelas (o *pmakes*) en donde los múltiples ficheros necesarios para crear un ejecutable se compilan y enlazan en paralelo.

5.4.2 Características de las cargas de trabajo

A continuación caracterizaremos algunas de las características básicas de nuestras cargas de trabajo, incluyendo la descomposición de los accesos a los datos en lecturas y escrituras o compartidos y privados, la concurrencia y la distribución de carga inherente, el ratio comunicación/cálculo y cómo se escala, y el tamaño y el escalamiento de los conjuntos de trabajo importantes. Las características relacionadas con la localidad espacial se miden en el contexto de los estilos de arquitectura específicos en capítulos posteriores. En esta sección, presentamos una caracterización cuantitativa de los datos para ejecuciones con 16 procesadores para nuestras aplicaciones paralelas y con 8 procesadores para el caso de carga de trabajo multiprogramada.

Acceso a los datos y características de sincronización

La tabla 5.1 resume los datos básicos y la frecuencia de los eventos de sincronización (cerrojos y barreras globales) en las diferentes cargas de trabajo. El conjunto de datos de entrada son los tamaños de problema por defecto, a no ser que se diga lo contrario. Los tamaños de problemas elegidos son lo bastante grandes para se de interés práctico para una máquina de hasta 64 procesadores pero lo suficientemente pequeño para ser simulados en un tiempo razonable.

Se ha seguido el comportamiento y las estadísticas de tiempo una vez que los procesos hijos han sido creados por el padre. Las referencias previas (realizadas por el procesador principal) se simulan pero no se incluyen en las estadísticas. En la mayoría de las aplicaciones, las medidas comienzan exactamente cuando se crean los procesos hijos. Las excepciones son Ocean y Barnes-Hut. En ambos casos, hemos hecho uso de la posibilidad de reducir drásticamente el número de pasos de tiempo para realizar la simulación (como se discutió en la sección 5.3.2). En este caso se han simulado un pequeño número de pasos de tiempo —seis para Ocean y cinco para Barnes-Hut— y se ha empezado a seguir el comportamiento y las estadísticas de tiempo después de los dos primeros pasos de tiempo. Para la carga de trabajo multiprogramada, las estadísticas se obtienen de un punto de chequeo tomado cerca del comienzo de *pmake*. Mientras que para todas las demás aplicaciones consideramos únicamente las

Application	**Data Set**	**Total Instr (M)**	**Total Flops (M)**	**Total Refs (M)**	**Total Reads (M)**	**Total Writes (M)**	**Shared Reads (M)**	**Shared Writes (M)**	**Bar- riers**	**Locks**
LU	**512x512 matrix 16x16 blocks**	489.52	92.20	151.07	103.09	47.99	92.79	44.74	66	0
Ocean	**258x258 grids tolerance=10^{-7} 4 time-steps**	376.51	101.54	99.70	81.16	18.54	76.95	16.97	364	1296
Barnes	**16K particles $\theta = 1.0$ 3 time-steps**	2002.74	239.24	720.13	406.84	313.29	225.04	93.23	7	34516
Radix	**256K points radix = 1024**	84.62	-	14.19	7.81	6.38	3.61	2.18	11	16
RayTrace	**Car scene**	833.35	-	290.35	210.03	80.31	161.10	22.35	0	94456
Radiosity	**Room scene**	2297.19	-	769.56	486.84	282.72	249.67	21.88	10	210485
Multiprog: User	**SGI Irix 5.2, two pmakes + two compress jobs**	1296.43	-	500.22	350.42	149.80	-	-	-	-
Multiprog: Kernel		668.10	-	212.58	178.14	34.44	-	-	-	621505

Tabla 5.1: Estadísticas generales acerca de los programas de aplicación.

referencias a los datos de las aplicaciones, para la carga de trabajo *Multiprog* también consideramos el impacto de las referencias de las instrucciones.

Concurrencia y equilibrado de la carga

Caracterizamos el equilibrio en la distribución de la carga midiendo el speedup algorítmico —es decir, el speedup sobre el modelo de arquitectura PRAM que asume que el acceso a los datos y las comunicaciones tienen una latencia cero (el coste de la instrucción que emite la referencia). Las desviaciones sobre el speedup ideal son atribuibles a desequilibrios en la carga, serialización en secciones críticas y el trabajo extra debido a los cálculos redundantes y manejo del paralelismo.

La figura 5.23 muestra los speedups algorítmicos para las seis aplicaiones y para 64 procesadores con el conjunto de datos por defecto. Tres de los programas (Barnes-Hut, Ocean y en menos extensión Raytrace) mejoran su rendimiento de forma casi lineal hasta 64 procesadores incluso con conjuntos de datos relativamente pequeños. Las fases dominantes de estos programas se pueden realizar en paralelo sobre un gran conjunto de datos (todas las partículas en Barnes-Hut, una malla completa en Ocean y la imagen de pixels en Raytrace). Únicamente presentan un paralelismo limitado y serialización en algunas operaciones de reducción globales y en porciones de algunas fases particulares que no son dominantes en términos de número de instrucciones ejecutadas (p.e., la creación del árbol y cerca de la raíz en la fase ascendente en Barnes, y en los niveles superiores de la jerarquía multimalla en Ocean). Raytrace presenta una sección crítica con algunos problemas de grave contención, causando serialización (de hecho, esta sección crítica no es estrictamente necesaria para la ejecución correcta pero se usa para mantener la pista de algunas estadísticas importantes).

Los seis programas muestran unos speedups algorítmicos buenos para hasta 16 ó 32 procesadores. Los programas que no se comportan muy bien para valores grandes de número de procesos con estos datos son LU, Radiosity y Radix. En cada caso, esto se debe al tamaño de los conjuntos de entrada más que

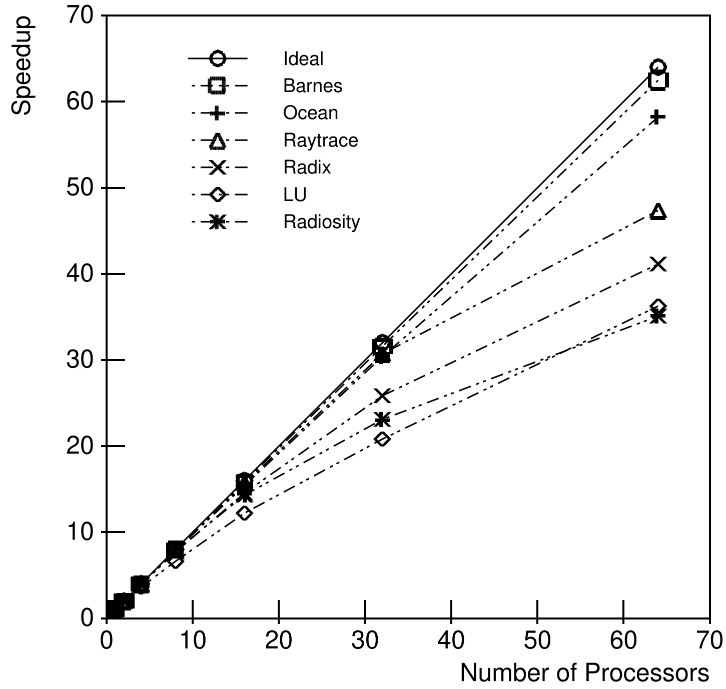


Figura 5.23: Speedups algorítmico para las seis aplicaciones paralelas.

a la naturaleza inherente de desequilibrio de las aplicaciones. En LU, el conjunto de datos por defecto da lugar un importante desequilibrio en la distribución de la carga para 64 procesadores, a pesar de la descomposición en bloques. Conjuntos de datos más grandes (o menos procesadores) reducen este desequilibrio proporcionando más bloques por procesador en cada paso de la factorización. Para Radiosity, el desequilibrio se debe también al uso de pequeños conjuntos de datos, aunque es muy difícil de analizar. Finalmente, para Radix el pobre speedup para 64 procesadores se debe al cálculo previo cuando se acumulan histogramas locales en un histograma global, que no puede paralelizarse completamente. El tiempo utilizado en este cálculo es $O(\log p)$, mientras que el tiempo utilizado en las otras fases es $O(\frac{n}{p})$, por lo que la fracción de trabajo total de esta fase no balanceada decrece cuando el número de claves a ordenar se incrementa. Así, incluso estos tres programas pueden usarse para evaluar grandes máquinas si elegimos conjuntos de datos mayores.

Ratio comunicación/cálculo

Incluimos en el ratio comunicación/cálculo la comunicación inherente al igual que la comunicación debida a la primera vez que un procesador accede a una palabra, si ocurre que este dato no está almacenado localmente (p.e., los fallos en frío que ocurren cuando comienza la medición). Donde sea posible, los datos están distribuidos de forma apropiada entre las memorias físicamente distribuidas, por lo que podemos considerar esta comunicación en frío es fundamental más que artificiosa. Para evitar la comunicación artificiosa debido a la capacidad finita o a la pobre localidad espacial, se han simulado cachés infinitas por procesador y bloques de cachés de una palabra. Medimos el ratio comunicación/cálculo como el número de bytes de datos de la aplicación que se comunican por instrucción, promediado por el número de procesadores. Para las aplicaciones intensivas en punto flotante (LU y Ocean), usamos bytes por FLOP en lugar de por instrucción dado que el número de FLOPs es menos sensible a las variaciones del compilador que el número total de instrucciones.

Aplicación	**Growth Rate of Comm/Comp Ratio**
LU	$\sqrt{P}$ / $\sqrt{DS}$
Ocean	$\sqrt{P}$ / $\sqrt{DS}$
Barnes	approximately $\sqrt{P}$ / $\sqrt{DS}$
Radiosity	unpredictable
Radix	(P-1)/P
Raytrace	unpredictable

Tabla 5.2: Ratios de crecimiento del ratio inherente entre comunicación y cálculo.

En primer lugar veamos el ratio medido de comunicación/cálculo para el tamaño de problema base que se muestra en la tabla 5.1. Esta tabla muestra como el ratio se incrementa con el número de procesadores bajo un escalamiento de tamaño del problema fijo. Posteriormente, cuando sea posible, examinaremos analíticamente la dependencia de este ratio del tamaño del conjunto de datos y del número de procesadores (Tabla 5.2) Los efectos de otros parámetros de la aplicación sobre el ratio comunicación/cálculo se discuten separadamente y, normalmente, de forma cualitativa.

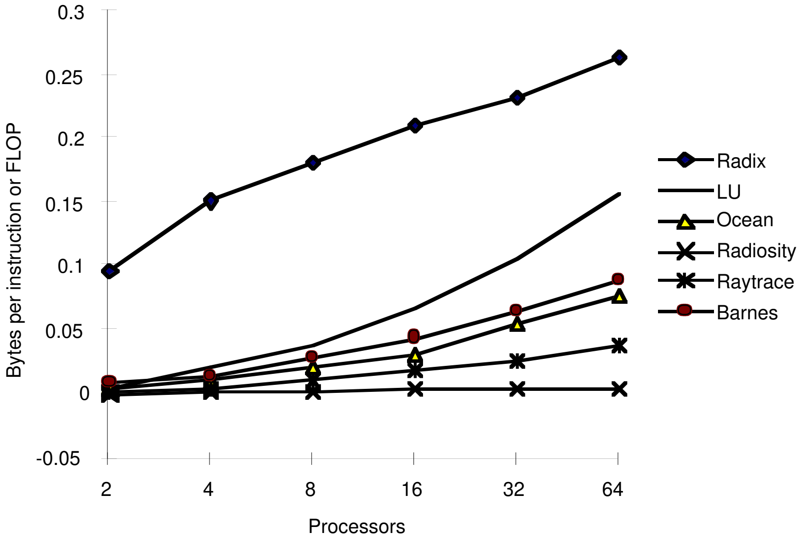


Figura 5.24: Ratio comunicación/cálculo en función del número de procesadores para el problema de tamaño base en las seis aplicaciones paralelas.

La figura 5.24 muestra los resultados medidos para el tamaño de problema base para nuestras seis aplicaciones paralelas. La primera cosa que observamos es que los ratios de comunicación/cálculo inherente medios son bastantes pequeños. Con procesadores operando a 400 MIPS, un ratio de 0.1 byte por instrucción es aproximadamente 40 MB/s de tráfico de datos, lo que es bastante pequeño para las modernas redes multiprocesador de alto rendimiento. El tráfico real es mucho más grande que el inherente debido a la comunicación artificiosa y a la información de control que se envía junto con los datos en cada transferencia. Esto indica que son los picos en la comunicación, las otras fuentes de comunicación y el patrón de la comunicación las causas probables de los problemas de ancho de banda en la comunicación, si éstos existen. La única aplicación cuyo ratio medio es bastante alto es

Radix, por lo que para esta aplicación es importante modelizar cuidadosamente el ancho de banda de las comunicaciones en las evaluaciones. Una razón de estos bajos ratios de comunicación/cálculo es que estamos utilizando aplicaciones muy bien optimizadas en sus asignaciones en la ejecución en paralelo. Las aplicaciones utilizadas en la práctica, incluyendo otras versiones de estas aplicaciones, pueden mostrar ratios mayores de comunicación/cálculo.

La siguiente observación que extraemos de la figura es que las tasas de crecimiento de los ratios comunicación/cálculo difieren en función de la aplicación, indicando un buen cubrimiento de esta propiedad. Estas tasas de crecimiento en función del número de procesadores y del tamaño del conjunto de datos (no mostrado en la figura) se resumen analíticamente en la tabla 5.2. El ratio comunicación/cálculo cambiaría dramáticamente en algunas aplicaciones (p.e., Ocean) si hubiésemos usado un tamaño para el conjunto de datos diferente, pero al menos la comunicación inherente no cambiaría demasiado para otras. La comunicación artificial es otra historia, y examinaremos el tráfico debido a la misma en el contexto de los diferentes tipos de arquitectura en capítulos posteriores.

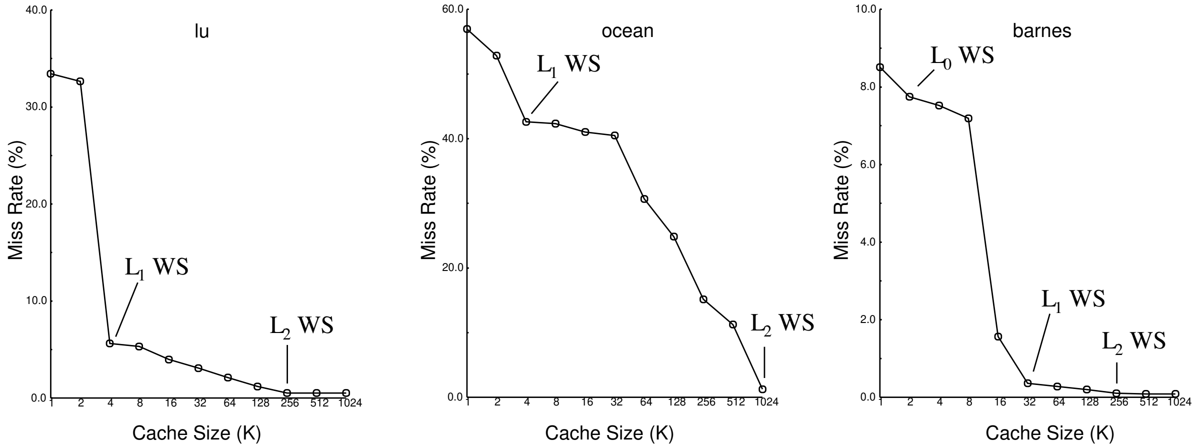
Mientras que las tasas de crecimiento son claramente fundamentales, es importante darse cuenta que no revelan las factores constantes de las expresiones de la razón comunicación/cálculo, que pueden ser en la práctica más importantes que las tasas de crecimiento asintóticas. Por ejemplo, si el ratio de un programa se incrementa con el logaritmo del número de procesadores, entonces asintóticamente su ratio será menor que aquél de una aplicación cuyo ratio crece según la raíz cuadrada del número de procesadores; sin embargo, puede ser realmente mucho mayor para todos los tamaños de máquina razonables si los factores constantes son mucho mayores. Los factores constantes de nuestras aplicaciones se pueden determinar a partir de la figura 5.24.

Tamaño de los conjuntos de trabajo

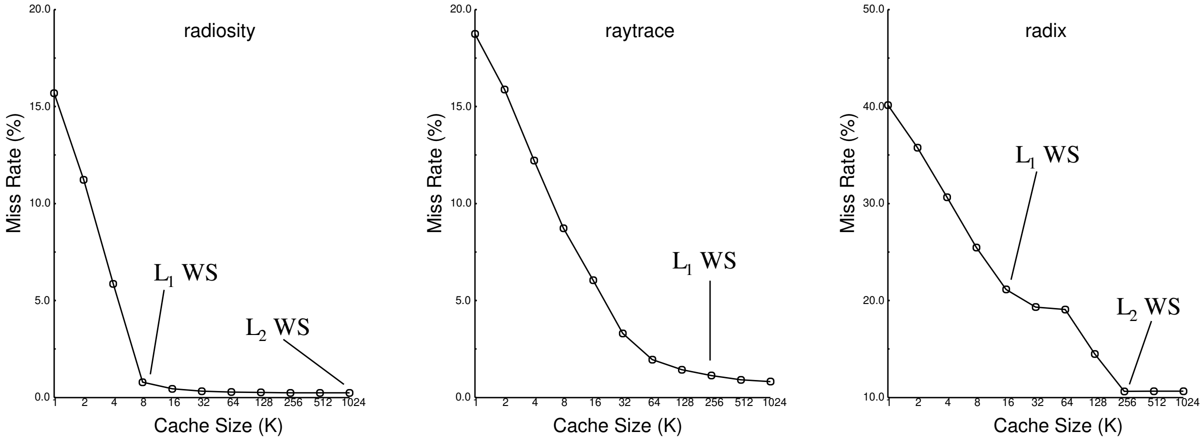
El tamaño de los conjuntos de trabajo de interés de un programa se puede medir más fácilmente utilizando cachés totalmente asociativas y con un tamaño de bloque de una palabra y simulando el programa con diferentes tamaños de caché para encontrar los codos en la curva de la tasa de fallo en función del tamaño de la caché. Asociatividades menores pueden hacer que sean necesarios mayores tamaños de la caché que los inherentes al tamaño del conjunto de trabajo, al igual que si usamos bloques de caché de más de una palabra (debido a la fragmentación en la caché).

La figura 5.25 muestra el resultado de las curvas de la tasa de fallo en función del tamaño de la caché para las seis aplicaciones paralelas, con los conjuntos de trabajo etiquetados como: conjunto de trabajo de nivel 1 (L_1 WS), conjunto de trabajo de nivel 2 (L_2 WS) y así sucesivamente. Una aplicación como Ocean tiene varios conjuntos de trabajo, pero nos centraremos en los dos definidos más abruptamente y en los importantes. Además de estos conjuntos de trabajo, algunas aplicaciones tienen pequeños conjuntos de trabajo que no varían con el tamaño del problema o el número de procesadores y que, por tanto, se espera que siempre estén en la caché, incluso en la más cercana al procesador; a éstos les denominaremos conjuntos de trabajo de nivel 0 (L_0 WS). Consisten, normalmente, en una pila de datos que el programa usa como almacenamiento temporal de cálculos primitivos (como la interacción de una celda partícula en Barnes-Hut) y que se reutiliza a lo largo de estos cálculos.

Podemos observar que en la mayoría de los casos los conjuntos de trabajo están muy definidos. La tabla 5.3 resume para los diferentes conjuntos de trabajo cómo varían sus tamaños con los parámetros de la aplicación y el número de procesadores, si son importantes para el rendimiento (al menos en máquinas con coherencia de caché eficiente), y si se puede esperar que no quepan en las cachés de segundo nivel para tamaños de problema reales (con un grado razonable de asociatividad de la caché).



(a) LU (b) Ocean (c) Barnes



(d) Radiosity (e) Raytrace (f) Radix

Figura 5.25: Curvas del conjunto de trabajo para las seis aplicaciones paralelas en ejecuciones con 16 procesadores.

Las aplicaciones para las que un conjunto de trabajo tiene un “Yes” en cada una de las dos últimas columnas son Ocean, Radix y Raytrace. Recordad que en Ocean, todos los cálculos más importantes

Program	**Working Set 1**	**Growth Rate**	**Important?**	**Realistic to not fit in cache?**	**Working Set 2**	**Growth Rate**	**Important?**	**Realistic to not fit in cache?**
LU	One block	Fixed(B)	Yes	No	partition of DS	DS/P	No	Yes
Ocean	A few subrows	$\sqrt{P}/\sqrt{DS}$	Yes	No	partition of DS	DS/P	Yes	Yes
Barnes	Tree data for 1 body	$(\log DS)/\theta^2$	Yes	No	partition of DS	DS/P	No	Yes
Radiosity	BSP tree	$\log(polygons)$	Yes	No	Unstructured	Unstructured	No	Yes
Radix	Histogram	Radix r	Yes	No	partition of DS	DS/P	Yes	Yes
Raytrace	Unstructured	Unstructured	Yes	No	Unstructured	Unstructured	Yes	Yes

Tabla 5.3: Conjuntos de trabajo importantes y sus tasas de crecimiento para el SPLASH-2.

5.5 Conclusiones finales

Ahora tenemos un buen entendimiento de los aspectos principales de la evaluación basada en carga de trabajo para los multiprocesadores: elección de las cargas de trabajo, escalamiento de los problemas y máquinas, manejo del gran espacio de parámetros y elección de las métricas. Para cada aspecto, tenemos un conjunto de líneas guía y pasos a seguir, un conocimiento de como evitar los fallos y un medio para entender las limitaciones en las investigaciones. También tenemos una base para nuestra propia ilustración de los compromisos arquitecturales en el resto de la asignatura.

Hemos visto que las cargas de trabajo deben elegirse para representar un amplio abanico de aplicaciones, patrones de comportamiento, y niveles de optimización. Mientras que las aplicaciones completas y, quizás, las cargas de trabajo multiprogramadas son indispensables, los microbenchmarks y kernels también juegan su papel.

También hemos visto que una evaluación adecuada mediante cargas de trabajo requiere un entendimiento de las propiedades relevantes de las cargas de trabajo al mismo tiempo que de sus interacciones con los parámetros de la arquitectura. Aunque este problema es complejo, hemos examinado una guía para tratar con espacios de parámetros grandes y reducirlos a la vez que se siguen cubriendo las situaciones reales.

La importancia de entender las propiedades relevantes de las cargas de trabajo se acentúa con el tema del escalamiento, que afecta a todas las características e interacciones importantes. Tanto el tiempo de ejecución como la memoria puede limitarse en el escalamiento, teniendo las aplicaciones a menudo más de un parámetro que determina las propiedades claves de la ejecución. Debemos escalar los programas basándonos en el entendimiento de estos parámetros, sus relaciones y sus impactos en el tiempo de ejecución y necesidades de memoria. Hemos visto que los modelos de escalamiento realistas dirigidos por las necesidades de las aplicaciones dan lugar a resultados de muy diferente significado arquitectónico. De hecho, el escalamiento es importante no sólo en la evaluación sino en el diseño.

Finalmente, hemos descrito las principales cargas de trabajo que usaremos como ilustración de la evaluación basada en cargas de trabajo de los sistemas reales de memoria compartida. (Para los sistemas de paso de mensajes, uno de las suites más utilizadas es la NAS Parallel Benchmarks II).